EP-36977

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04J 1/02 — A2

(11) International Publication Number: **WO 98/19410**

(43) International Publication Date: 7 May 1998 (07.05.98)

(21) International Application Number: PCT/US97/18911

(22) International Filing Date: 22 October 1997 (22.10.97)

(30) Priority Data:

9622728.5	31 October 1996 (31.10.96)	GB
9720550.4	26 September 1997 (26.09.97)	GB

(81) Designated States: AL, AM, AT, AU, AZ, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published

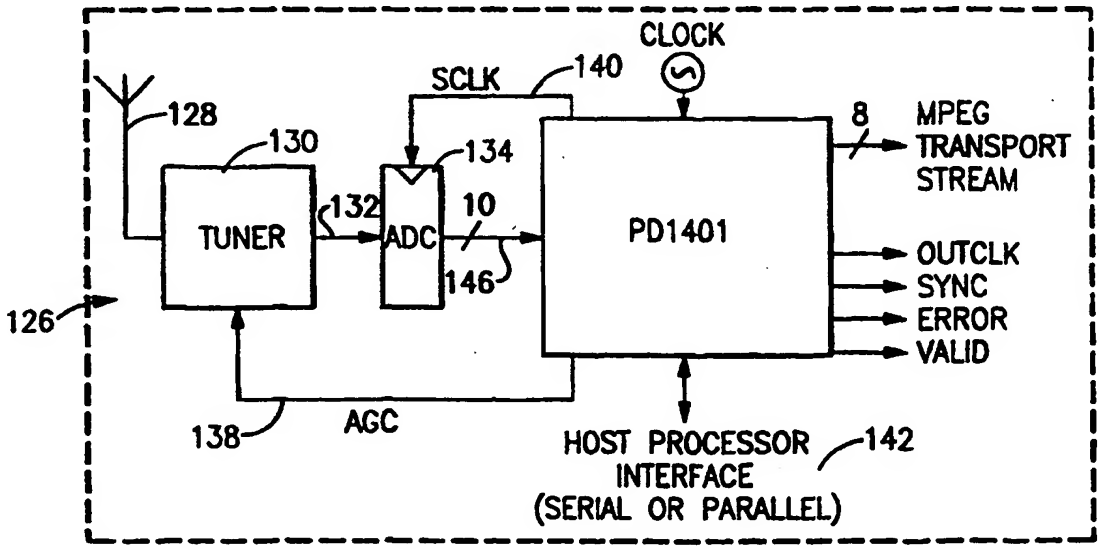
Without international search report and to be republished upon receipt of that report.

(71) Applicant: DISCOVISION ASSOCIATES [US/US]; Suite 200, 2355 Main Street, P.O. Box 19616, Irvine, CA 92623 (US).

(72) Inventors: ALAM, Dawood; 15 Westbury Park, Durdham Down, Bristol BS6 7JA (GB). COLLINS, Matthew, James; Flat 1, 4 New King Street, Bath BA1 2BN (GB). DAVIES, David, Huw; 6 Glen Brook, Glen Drive, Stoke Bishop, Bristol BS9 1SB (GB). KEEVILL, Peter, Anthony; 7 Junction Road, Oldfield Park, Bath BA2 3NQ (GB). NOLAN, John, Matthew; 19 The Firs, Combe Down, Bath, Somerset BA2 5ED (GB). FOXCROFT, Thomas; 52B Pembroke Road, Clifton, Bristol BS8 3DT (GB). PARKER, Jonathan; 66 Third Avenue, Oldfield Park, Bath BA2 3NZ (GB).

(74) Agent: BICKEL, Arthur, S.; Suite 200, 2355 Main Street, P.O. Box 19616, Irvine, CA 92623 (US).

(54) Title: SINGLE CHIP VLSI IMPLEMENTATION OF A DIGITAL RECEIVER EMPLOYING ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING



(57) Abstract

The invention provides a single chip implementation of a digital receiver for multicarrier signals that are transmitted by orthogonal frequency division multiplexing. Improved channel estimation and correction circuitry are provided. The receiver has highly accurate sampling rate control and frequecy control circuitry. BCH decoding of tps data carriers is achieved with minimal resources with an arrangement that includes a small Galois field multiplier. An improved FFT window synchronization circuit is coupled to the resampling circuit for locating the boundary of the guard interval transmitted with the active frame of the signal. A real–time pipelined FFT processor is operationally associated with the FFT window synchronization circuit and operates with reduced memory requirements.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

SINGLE CHIP VLSI IMPLEMENTATION OF A DIGITAL RECEIVER EMPLOYING ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING

This invention relates to receivers of electromagnetic signals employing multicarrier modulation. More particularly this invention relates to a digital receiver which is implemented on a single VLSI chip for receiving transmissions employing orthogonal frequency division multiplexing, and which is suitable for the reception of digital video broadcasts.

Coded orthogonal frequency division multiplexing ("COFDM") has been proposed for digital audio and digital video broadcasting, both of which require efficient use of limited bandwidth, and a method of transmission which is reliable in the face of several effects. For example the impulse response of a typical channel can be modeled as the sum of a plurality of Dirac pulses having different delays. Each pulse is subject to a multiplication factor, in which the amplitude generally follows a Rayleigh law. Such a pulse train can extend over several microseconds, making unencoded transmission at high bit rates unreliable. In addition to random noise, impulse noise, and fading, other major difficulties in digital terrestrial transmissions at high data rates include multipath propagation, and adjacent channel interference, where the nearby frequencies have highly correlated signal variations. COFDM is particularly suitable for these applications. In practical COFDM arrangements, relatively small amounts of data are modulated onto each of a large number of carriers that are closely spaced in frequency. The duration of a data symbol is increased in the same ratio as the number of carriers or subchannels, so that inter-symbol interference is markedly reduced.

Multiplexing according to COFDM is illustrated in Figs. 1 and 2, wherein the spectrum of a single COFDM carrier or subchannel is indicated by line 2. A set of carrier frequencies is indicated by the superimposed waveforms in Fig. 2, where orthogonality conditions are satisfied. In general two real-valued functions are orthogonal if

$$\int_a^b \psi_p(t)\,\psi_q^*(t)\,dt = K \qquad (1)$$

where K is a constant, and K = 0 if $p \neq q$; $K \neq 0$ if p = q. Practical encoding and decoding of signals according to COFDM relies heavily on the fast Fourier transform ("FFT"), as can be appreciated from the following equations.

The signal of a carrier c is given by

$$s_c(t) = A_c(t)\,e^{j[\omega_c t + \phi_c(t)]} \qquad (2)$$

where A_c is the data at time t, ω_c is the frequency of the carrier, and ϕ_c is the phase. N carriers in the COFDM signal is given by

$$s_s(t) = (1/N) \sum_{n=0}^{N} A_n(t) e^{j[\omega_n t + \phi_n(t)]} \quad (3)$$

$$\omega_n = \omega_0 + n\Delta\omega \quad (4)$$

Sampling over one symbol period, then

$$\phi_c(t) \Rightarrow \phi_n \quad (5)$$

$$A_c(t) \Rightarrow A_n \quad (6)$$

With a sampling frequency of 1/T, the resulting signal is represented by

$$s_s(t) = (1/N) \sum_{n=0}^{N} A_n(t) e^{j[(\omega_n + n\Delta\omega)kT + \phi_n]} \quad (7)$$

Sampling over the period of one data symbol $\tau = NT$, with $\omega_0 = 0$,

$$s_s(kT) = (1/N) \sum_{n=0}^{N-1} A_n e^{j\phi_n} e^{j(n\Delta\omega)kT} \quad (8)$$

which compares with the general form of the inverse discrete Fourier transform:

$$g(kT) = (1/N) \sum_{n=0}^{N-1} G(n/(kT)) e^{j\pi n(k/N)} \quad (9)$$

In the above equations $A_n e^{j\phi}{}_n$ is the input signal in the sampled frequency domain, and $s_s(kT)$ is the time domain representation. It is known that increasing the size of the FFT provides longer symbol durations and improves ruggedness of the system as regards echoes which exceed the length of the guard interval. However computational complexity increases according to $N\log_2 N$, and is a practical limitation.

In the presence of intersymbol interference caused by the transmission channel, orthogonality between the signals is not maintained. One approach to this problem has been to deliberately sacrifice some of the emitted energy by preceding each symbol in the time domain by an interval which exceeds the memory of the channel, and any multipath delay. The "guard interval" so chosen is large enough to absorb any intersymbol interference, and is established by preceding each symbol by a replication of a portion of itself. The replication is typically a cyclic extension of the terminal portion

of the symbol. Referring to Fig. 3, a data symbol 4 has an active interval 6 which contains all the data transmitted in the symbol. The terminal portion 8 of the active interval 6 is repeated at the beginning of the symbol as the guard interval 10. The COFDM signal is represented by the solid line 12. It is possible to cyclically repeat initial portion of the active interval 6 at the end of the symbol.

Transmission of COFDM data can be accomplished according to the known general scheme shown in Fig. 4. A serial data stream 14 is converted to a series of parallel streams 16 in a serial-to-parallel converter 18. Each of the parallel streams 16 is grouped into x bits each to form a complex number, where x determines the signal constellation of its associated parallel stream. After outer coding and interleaving in block 20 pilot carriers are inserted via a signal mapper 22 for use in synchronization and channel estimation in the receiver. The pilot carriers are typically of two types. Continual pilot carriers are transmitted in the same location in each symbol, with the same phase and amplitude. In the receiver, these are utilized for phase noise cancellation, automatic frequency control, and time/sampling synchronization. Scattered pilot carriers are distributed throughout the symbol, and their location typically changes from symbol to symbol. They are primarily useful in channel estimation. Next the complex numbers are modulated at baseband by the inverse fast fourier transform ("IFFT") in block 24. A guard interval is then inserted at block 26. The discrete symbols are then converted to analog, typically low-pass filtered, and then upconverted to radiofrequency in block 28. The signal is then transmitted through a channel 30 and received in a receiver 32. As is well known in the art, the receiver applies an inverse of the transmission process to obtain the transmitted information. In particular an FFT is applied to demodulate the signal.

A modern application of COFDM has been proposed in the European Telecommunications Standard ETS 300 744 (March 1997), which specifies the framing structure, channel coding, and modulation for digital terrestrial television. The specification was designed to accommodate digital terrestrial television within the existing spectrum allocation for analog transmissions, yet provide adequate protection against high levels of co-channel interference and adjacent channel interference. A flexible guard interval is specified, so that the system can support diverse network configurations, while maintaining high spectral efficiency, and sufficient protection against co-channel interference and adjacent channel interference from existing PAL/SECAM services. The noted European Telecommunications Standard defines two modes of operation. A "2K mode", suitable for single transmitter operation and for small single frequency networks with limited transmitter distances. An "8K mode" can be used for either single transmitter operation or for large single frequency networks. Various levels of quadrature amplitude

modulation ("QAM") are supported, as are different inner code rates, in order to balance bit rate against ruggedness. The system is intended to accommodate a transport layer according to the Moving Picture Experts Group ("MPEG"), and is directly compatible with MPEG-2 coded TV signals (ISO/IEC 13818).

In the noted European Telecommunications Standard data carriers in a COFDM frame can be either quadrature phase shift keyed ("QPSK"), 16-QAM, 64-QAM, non-uniform 16-QAM, or non-uniform 64-QAM using Gray mapping.

An important problem in the reception of COFDM transmission is difficulty in maintaining synchronization due to phase noise and jitter which arise from upconversion prior to transmission, downconversion in the receiver, and the front end oscillator in the tuner, which is typically a voltage controlled oscillator. Except for provision of pilot carriers to aid in synchronization during demodulation, these issues are not specifically addressed in the noted European Telecommunications Standard, but are left for the implementer to solve.

Basically phase disturbances are of two types. First, noisy components which disturb neighbor carriers in a multicarrier system are called the "foreign noise contribution" ("FNC"). Second, a noisy component which disturbs its own carrier is referred to as the "own noise contribution".

Referring to Fig. 5, the position of ideal constellation samples are indicated by "x" symbols 34. The effect of foreign noise contribution is stochastic, resulting in Gaussian-like noise. Samples perturbed in this manner are indicated on Fig. 5 as circles 36. The effects of the own noise contribution is a common rotation of all constellation points, indicated as a displacement between each "x" symbol 34 and its associated circle 36. This is referred to as the "common phase error", which notably changes from symbol to symbol, and must therefore be recalculated each symbol period T_S. The common phase error may also be interpreted as a mean phase deviation during the symbol period T_S.

In order for the receiver 32 to process the data symbols in a practical system, a mathematical operation is performed on the complex signal representing each data symbol. Generally this is an FFT. For valid results to be obtained, a particular form of timing synchronization is required in order to align the FFT interval with the received data symbol.

It is therefore a primary object of the invention to provide a highly integrated, low cost apparatus for the reception of digital broadcasts, such as terrestrial digital video broadcasts, which is implemented on a single VLSI chip.

It is another object of the invention to provide an improved method and apparatus for synchronizing a received data symbol with an FFT window in signals transmitted according to COFDM.

It is yet another object of the invention to improve the stability of digital multicarrier receivers in respect of channel estimation.

It is still another object of the invention to improve the automatic frequency control circuitry employed in multicarrier digital receivers.

It is a further object of the invention to improve the automatic sampling rate control circuitry employed in multicarrier digital receivers.

The invention provides a digital receiver for multicarrier signals that are transmitted by orthogonal frequency division multiplexing. The multicarrier signal carries a stream of data symbols having an active interval, and a guard interval in which the guard interval is a replication of a portion of the active interval. In the receiver an analog to digital converter is coupled to a front end amplifier. An I/Q demodulator is provided for recovering in phase and quadrature components from data sampled by the analog to digital converter, and an automatic gain control circuit is coupled to the analog to digital converter. In a low pass filter circuit accepting I and Q data from the I/Q demodulator, the I and Q data are decimated and provided to a resampling circuit. An interpolator in the resampling circuit accepts the decimated I and Q data at a first rate and outputs resampled I and Q data at a second rate. An FFT window synchronization circuit is coupled to the resampling circuit for locating a boundary of the guard interval. A real-time pipelined FFT processor is operationally associated with the FFT window synchronization circuit. Each stage of the FFT processor has a complex coefficient multiplier, and an associated memory with a lookup table defined therein for multiplicands being multiplied in the complex coefficient multiplier. Each multiplicand in the lookup table is unique in value. A monitor circuit responsive to the FFT window synchronization circuit detects a predetermined indication that a boundary between an active symbol and a guard interval has been located.

According to an aspect of the invention the FFT window synchronization circuit has a first delay element accepting currently arriving resampled I and Q data, and outputting delayed resampled I and Q data. A subtracter produces a signal representative of the difference between the currently arriving resampled I and Q data and the delayed resampled I and Q data. In a first circuit the subtracter output signal is converted to a signal having a unipolar magnitude, which is preferably the absolute value of the signal provided by the subtracter. A second delay element stores the output signal of the first circuit, and a third delay element receives the delayed output of the second delay element. In a second circuit a statistical relationship is calculated between data stored in the second delay element and data stored in the third delay element. The output of the FFT window synchronization circuit is representative of the statistical relationship.

Preferably the statistical relationship is the F ratio. The FFT processor is capable of operation in a 2K mode and in an 8K mode.

The FFT processor has an address generator for the memory of each stage, which accepts a signal representing the order dependency of a currently required multiplicand, and generates an address of the memory wherein the currently required multiplicand is stored. In a further aspect of the invention each multiplicand is stored in the lookup table in order of its respective order dependency for multiplication by the complex coefficient multiplier, so that the order dependencies of the multiplicands define an incrementation sequence. The address generator has an accumulator for storing a previous address that was generated thereby, a circuit for calculating an incrementation value of the currently required multiplicand responsive to the incrementation sequence, and an adder for adding the incrementation value to the previous address.

In another aspect of the invention there are a plurality of incrementation sequences. The multiplicands are stored in row order, wherein in a first row a first incrementation sequence is 0, in a second row a second incrementation sequence is 1, in a third row first and second break points B1, B2 of a third incrementation sequence are respectively determined by the relationships

$$B1_{M_N} = 4^N B1_{M_N} - \sum_{n=0}^{N-1} 4^n$$

$$B2_{M_N} = \sum_{n=0}^{N} 4^n$$

and in a fourth row a third break point B3 of a third incrementation sequence is determined by the relationship

$$B3_{M_N} = 2 \times 4^N + 2$$

wherein M_N represents the memory of an Nth stage of the FFT processor.

The receiver provides channel estimation and correction circuitry. Pilot location circuitry receives a transformed digital signal representing a frame from the FFT processor, and identifies the position of pilot carriers therein. The pilot carriers are spaced apart in a carrier spectrum of the transformed digital signal at intervals K and have predetermined magnitudes. The pilot location circuitry has a first circuit for computing an order of carriers in the transformed digital signal, positions of said carriers being calculated modulo K. There are K accumulators coupled to the second circuit for accumulating magnitudes of the carriers in the transformed digital signal, the accumulated magnitudes defining a set. A correlation circuit is provided for correlating

K sets of accumulated magnitude values with the predetermined magnitudes. In the correlation a first member having a position calculated modulo K in of each of the K sets is uniquely offset from a start position of the frame.

According to another aspect of the invention the pilot location circuitry also has a bit reversal circuit for reversing the bit order of the transformed digital signal.

According to yet another aspect of the invention amplitudes are used to represent the magnitudes of the carriers. Preferably the magnitudes of the carriers and the predetermined magnitudes are absolute values.

In a further aspect of the invention the correlation circuitry also has a peak tracking circuit for determining the spacing between a first peak and a second peak of the K sets of accumulated magnitudes, wherein the first peak is the maximum magnitude, and the second peak is the second highest magnitude.

The channel estimation and correction circuitry also has an interpolating filter for estimating the channel response between the pilot carriers, and a multiplication circuit for multiplying data carriers output by the FFT processor with a correction coefficient produced by the interpolating filter.

The channel estimation and correction circuitry also has a phase extraction circuit accepting a data stream of phase-uncorrected I and Q data from the FFT processor, and producing a signal representative of the phase angle of the uncorrected data. The phase extraction circuit includes an accumulator for the phase angles of succeeding phase-uncorrected I and Q data.

According to an aspect of the invention the channel estimation and correction circuitry includes an automatic frequency control circuit coupled to the phase extraction circuit, in which a memory stores the accumulated common phase error of a first symbol carried in the phase-uncorrected I and Q data. An accumulator is coupled to the memory and accumulates differences between the common phase error of a plurality of pilot carriers in a second symbol and the common phase error of corresponding pilot carriers in the first symbol. The output of the accumulator is filtered, and coupled to the I/Q demodulator.

According to another aspect of the invention the coupled output of the accumulator of the automatic frequency control circuit is enabled in the I/Q demodulator only during reception of a guard interval therein.

According to yet another aspect of the invention the channel estimation and correction circuitry also has an automatic sampling rate control circuit coupled to the phase extraction circuit, in which a memory stores the individual accumulated phase errors of pilot carriers in a first symbol carried in the phase-uncorrected I and Q data. An accumulator is coupled to the memory and accumulates differences between the

phase errors of individual pilot carriers in a second symbol and phase errors of corresponding pilot carriers in the first symbol to define a plurality of accumulated intersymbol carrier phase error differentials. A phase slope is defined by a difference between a first accumulated intersymbol carrier phase differential and a second accumulated intersymbol carrier phase differential. The output of the accumulator is filtered and coupled to the I/Q demodulator.

According to one aspect of the invention the sampling rate control circuit stores a plurality of accumulated intersymbol carrier phase error differentials and computes a line of best fit therebetween.

According to another aspect of the invention the coupled output signal of the accumulator of the automatic sampling rate control circuit is enabled in the resampling circuit only during reception of a guard interval therein.

According to an aspect of the invention a common memory for storing output of the phase extraction circuit is coupled to the automatic frequency control circuit and to the automatic sampling rate control circuit.

According to another aspect of the invention the phase extraction circuit also has a pipelined circuit for iteratively computing the arctangent of an angle of rotation according to the series

$$\tan^{-1}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \ldots, \qquad |x| < 1$$

wherein x is a ratio of the phase-uncorrected I and Q data.

The pipelined circuit includes a constant coefficient multiplier, and a multiplexer for selecting one of a plurality of constant coefficients of the series. An output of the multiplexer is connected to an input of the constant coefficient multiplier.

According to still another aspect of the invention the pipelined circuit has a multiplier, a first memory for storing the quantity x^2, wherein the first memory is coupled to a first input of the multiplier, and has a second memory for holding an output of the multiplier. A feedback connection is provided between the second memory and a second input of the multiplier. The pipelined circuit also has a third memory for storing the value of the series. Under direction of a control circuit coupled to the third memory, the pipeline circuit computes N terms of the series, and also computes N+1 terms of the series. An averaging circuit is also coupled to the third memory and computes the average of N terms and N+1 terms of the series.

Data transmitted in a pilot carrier of the multicarrier signal is BCH encoded according to a code generator polynomial h(x). A demodulator operative on the BCH encoded data is provided, which includes an iterative pipelined BCH decoding circuit.

The BCH decoding circuit is circuit coupled to the demodulator. It forms a Galois Field of the polynomial, and calculates a plurality of syndromes therewith. The BCH decoding circuit includes a plurality of storage registers, each storing a respective one of the syndromes, and a plurality of feedback shift registers, each accepting data from a respective one of the storage registers. The BCH decoding circuit has a plurality of Galois field multipliers. Each of the multipliers is connected in a feedback loop across a respective one of the feedback shift registers and multiplies the output of its associated feedback shift register by an alpha value of the Galois Field. An output Galois field multiplier multiplies the outputs of two of the feedback shift registers.

A logical network forms an error detection circuit connected to the feedback shift registers and to the output Galois field multiplier. The output of the error detection circuit indicates an error in a current bit of data, and a feedback line is enabled by the error detection logic and connected to the storage registers. Using the feedback line, the data output by the feedback shift registers are written back into the storage registers for use in a second iteration.

According to an aspect of the invention the output Galois field multiplier has a first register initially storing a first multiplicand A, a constant coefficient multiplier connected to the first register for multiplication by a value α. An output of the constant coefficient multiplier is connected to the first register to define a first feedback loop, whereby in a kth cycle of clocked operation the first register contains a Galois field product $A\alpha^k$. A second register is provided for storing a second multiplicand B. An AND gate is connected to the second register and to the output of the constant coefficient multiplier. An adder has a first input connected to an output of the AND gate. An accumulator is connected to a second input of the adder, and the Galois field product AB is output by the adder.

The invention provides a method for the estimation of a frequency response of a channel. It is performed by receiving from a channel an analog multicarrier signal that has a plurality of data carriers and scattered pilot carriers. The scattered pilot carriers are spaced apart at an interval N and are transmitted at a power that differs from the transmitted power of the data carriers. The analog multicarrier signal is converted to a digital representation thereof. A Fourier transform is performed on the digital representation of the multicarrier signal to generate a transformed digital signal. The bit order of the transformed digital signal is reversed to generate a bit-order reversed signal. Magnitudes of the carriers in the bit-order reversed signal are cyclically accumulated in N accumulators, amd the accumulated magnitudes are correlated with the power of the scattered pilot carriers. Responsive to the correlation, a synchronizing signal is

generated that identifies a carrier position of the multicarrier signal, preferably an active carrier.

According to another aspect of the invention the step of accumulating magnitudes is performed by adding absolute values of a real component of the bit-order reversed signal to respective absolute values of imaginary components thereof to generate sums, and respectively storing the sums in the N accumulators.

According to yet another aspect of the invention the step of correlating the accumulated magnitudes also is performed by identifying a first accumulator having the highest of the N values stored therein, which represents a first carrier position, and by identifying a second accumulator which has the second highest of the N values stored therein, which represents a second carrier position. The interval between the first carrier position and the second carrier position is then determined.

To validate the consistency of the carrier position identification, the position of a carrier of a first symbol in the bit-order reversed signal is compared with a position of a corresponding carrier of a second symbol therein.

Preferably interpolation is performed between pilot carriers to determine correction factors for respective intermediate data carriers disposed therebetween, and respectively adjusting magnitudes of the intermediate data carriers according to the correction factors.

According to an aspect of the invention a mean phase difference is determined between corresponding pilot carriers of successive symbols of the transformed digital signal. A first control signal representing the mean phase difference, is provided to control the frequency of reception of the multicarrier signal. The first control signal is enabled only during reception of a guard interval.

Preferably a line of best fit is determined for the inter-symbol phase differences of multiple carriers to define a phase slope.

For a better understanding of these and other objects of the present invention, reference is made to the detailed description of the invention, by way of example, which is to be read in conjunction with the following drawings, wherein:

Fig. 1 illustrates the spectrum of a COFDM subchannel;

Fig. 2 shows a frequency spectrum for multiple carriers in a COFDM signal;

Fig. 3 is a diagram of a signal according to COFDM and shows a data symbol format;

Fig. 4 is a block diagram illustrating an FFT based COFDM system;

Fig. 5 illustrates certain perturbations in a COFDM signal constellation;

Fig. 6 is a flow diagram of a method of timing synchronization according to a preferred embodiment of the invention;

Fig. 7 is a plot of an F ratio test performed on several data symbols for coarse timing synchronization;

Fig. 8 is a plot of an incomplete beta function for different degrees of freedom;

Fig. 9 is a plot helpful in understanding a test of statistical significance according to the invention;

Fig. 10 is an electrical schematic of a synchronization circuit according to an alternate embodiment of the invention;

Fig. 11 is an electrical schematic of a synchronization circuit according to another alternate embodiment of the invention;

Fig. 12 is a block diagram of a single-chip embodiment of a digital receiver in accordance with the invention;

Fig. 13 is a block diagram illustrating the front end of the digital receiver shown in Fig. 12 in further detail;

Fig. 14 is a block diagram illustrating the FFT circuitry, channel estimation and correction circuitry of the digital receiver shown in Fig. 12;

Fig. 15 is a block diagram illustrating another portion of the digital receiver shown in Fig. 12;

Fig. 16 is a more detailed block diagram of the channel estimation and correction circuitry shown in Fig. 14;

Fig. 17 is a schematic of the automatic gain control circuitry of the digital receiver shown in Fig. 12;

Fig. 18 is a schematic of the I/Q demodulator of the digital receiver shown in Fig. 12;

Fig. 19 illustrates in greater detail a low pass filter shown in Fig. 13;

Fig. 20 shows the response of the low pass filter shown in Fig. 19;

Fig. 21 shows the resampling circuitry of the digital receiver shown in Fig. 12;

Fig. 22 illustrates a portion of an interpolator in the resampling circuitry of Fig. 21;

Fig. 23 is a more detailed block diagram of the FFT window circuitry shown in Fig. 14;

Fig. 24 is a schematic of a butterfly unit in the FFT calculation circuitry shown in Fig.14;

Figs. 25 and 26 are schematics of butterfly units in accordance with the prior art;

Fig. 27 is a schematic of a radix 2^2 + 2 FFT processor in accordance with the invention;

Fig. 28 is 32 point flow graph of the FFT processor shown in Fig. 27;

Fig. 29 is a schematic of a configurable 2K/8K radix 2^2+2 single path, delay feedback pipelined FFT processor in accordance with the invention;

Fig. 30 is a detailed schematic of a complex multiplier used in the circuitry shown in Fig. 29;

Fig. 31 is a detailed schematic of an alternate embodiment of a complex multipliers used in the circuitry shown in Fig. 29;

Fig. 32 is another diagram illustrating the organization of the twiddle factors for each of the multipliers in the circuitry shown in Fig. 29;

Fig. 33 illustrates the organization of the twiddle factors for each of the multipliers in the circuitry shown in Fig. 29;

Fig. 34 is a schematic of address generator used in the circuitry shown in Fig. 29;

Fig. 35 is a schematic of a generalization of the address generator shown in Fig. 34;

Fig. 36 is a flow chart illustrating the process of pilot location conducted by the channel estimation and correction circuitry shown in Fig. 16;

Fig. 37 is a flow chart of an embodiment of the pilot localization procedure according to the invention.

Fig. 38 is a more detailed block diagram of the tps sequence block of the circuitry shown in Fig. 14;

Fig. 39 is a schematic of a BCH decoder used in the tps processing circuitry shown in Fig. 38;

Fig. 40 is a more detailed schematic of a Galois field multiplier shown in Fig. 39;

Fig. 41 is a block diagram generically illustrating the automatic sampling control and automatic frequency control loops of the digital receiver shown in Fig. 12;

Fig. 42 is a more detailed block diagram of the automatic sampling control and automatic frequency control loops shown in Fig. 41;

Fig. 43 is a more detailed block diagram of the phase extract block of the circuitry shown in Fig. 42;

Fig. 44 is a schematic of the circuitry employed to calculate an arctangent in the block diagram shown in Fig. 43;

Fig. 45 is a plot of the square error at different values of α of the Taylor expansion to 32 terms;

Fig. 46 is a plot of the square error at different values of α of the Taylor expansion to 31 terms;

Fig. 47 is a plot of the square error at different values of α of the average of the Taylor expansion to 31 and 32 terms;

Fig. 48 is a plot of the phase differences of pilot carriers with a line of best fit shown;

Fig. 49 is a more detailed block diagram an alternate embodiment of the automatic sampling control and automatic frequency control loops shown in Fig. 41;

Fig. 50 illustrates a coded constellation format used in the demapping circuitry of Fig. 15;

Fig. 51 illustrates the conversion of I,Q data to binary data value using the format shown in Fig. 50;

Fig. 52 is a more detailed block diagram of the symbol deinterleaving circuitry shown in Fig. 15;

Fig. 53 is a more detailed block diagram of the bit deinterleaving circuitry shown in Fig. 15;

Fig. 54 illustrates the conversion from a coded constellation format to a 24 bit soft I/Q format by the bit deinterleaving circuitry shown in Fig. 53;

Fig. 55 is a more detailed block diagram of the microprocessor interface of the receiver shown in Fig. 12;

Fig. 56 is a more detailed block diagram of the system controller of the receiver shown in Fig. 12; and

Fig. 57 is a state diagram relating to channel acquisition in the system controller of the receiver shown in Fig. 56.

Alignment of The FFT Window

Referring again to Figs. 3 and 4, according to the invention a statistical method is applied to COFDM signals to find the end of the guard interval 10. This method is explained with reference to the above noted European Telecommunications Standard, but is applicable to many forms of frequency division multiplexing having prefixed or postfixed guard intervals. It allows the receiver 32 to find the end of the guard interval given only the received sampled complex signal (solid line 12) and the size of the active interval 6 . The method relies on the fact that the guard interval 10 is a copy of the last part of the data symbol 4. In the receiver 32, due to echoes and noise from the channel and errors in the local oscillator, the guard interval 10 and the last part of the data symbol 4 will differ. If the errors introduced are random then a statistical method can be applied. According to the invention, the received complex signal is sampled at a rate which is nearly identical to that used in the transmitter. A difference signal is found for a pair of received samples which are separated by a period of time which is as close as possible to the active interval 6. This period should be equal to the size of the fast fourier transform ("FFT") being applied (i.e. 2048 or 8192 samples). Let

$$S_i = |s_i| - |s_{i-fftsize}| \qquad (14)$$

where S_i is the difference signal; s_i and $s_{i\text{-fftsize}}$ are the current and previous complex input samples of which the modulus is taken. That is, the subscript "i" indexes a linear time sequence of input values. Assuming that the input signal is random, then S_i is also random. Within the guard interval s_i and $s_{i\text{-fftsize}}$ will be similar, although not identical, due to the effects of the channel. S_i will be therefore a random signal with a small dispersion. As used herein the term "dispersion" means generally the spread of values, and is not restricted to a particular mathematical definition. In general the active part of one symbol is not related to the active part of the next symbol. Outside of the guard interval S_i will be random with a much larger dispersion. In order to find the end of the guard interval, the dispersion of the difference signal S_i is monitored to look for a significant increase which will occur at the boundary of the guard interval 10 and the active interval 6. The inventors have also observed that a large decrease in dispersion is seen at the start of the guard interval 10.

According to a preferred embodiment of the invention samples of the input signal are stored over an interval which includes at least one symbol period T_s. The dispersion of the difference signal S_i is calculated over a block of samples. The block is moved back in time over a number of samples, n, and the dispersion is recalculated. These two blocks are referred to herein as "comparison blocks". The ratio of a current dispersion in a first comparison block to the dispersion in a previous comparison block is found. Then, the F ratio significance test is used to find significant differences in the dispersions of the two comparison blocks. The F ratio is defined as

$$F=\frac{VAR(i)}{VAR(i-n)} \qquad (15)$$

where n is a positive integer, i indexes the input samples, and VAR(i) is the variance of a block of values of length N samples. Variance can be defined as

$$VAR(i)=\frac{1}{N}\sum_{j=0}^{N}(S_{i-j})^2-\left(\frac{1}{N}\sum_{j=0}^{N}S_{i-j}\right)^2 \qquad (16)$$

While the F ratio significance test is used in the preferred embodiment, other functions of the two dispersion values which give a signal relating to the change in dispersion could be used. There are many such functions. An advantage of the F ratio is that for a random input signal it has a known probability distribution, allowing convenient statistical analysis for purposes of performance analysis and system design. Also the F ratio intrinsically normalizes the signal, making the result independent of the signal level.

The method is disclosed with reference to Fig. 6, in which a first member of a sample pair in a current evaluation block is measured at step 38. A delay of one active interval 6 (Fig. 3) is experienced in step 40. This may be accomplished with a digital delay such as a FIFO, or equivalently by buffering samples for an active interval in a memory and accessing appropriate cells of the memory. A second member of the sample pair is measured in step 42, and the difference between the first and second member is determined and stored in step 44. The end of the current block is tested at decision step 46. The size of the evaluation block should not exceed the length of a guard interval, and may be considerably smaller. In the event the end of the current block has not yet been reached, another sample is acquired at step 48, and control returns to step 38.

If the end of the current block has been reached, the dispersion of the current block is measured in step 50, and is treated as one of two comparison blocks of data. A test is made at decision step 52 to determine if a group of two comparison blocks have been evaluated. If this test is negative, then another block of data is acquired in step 54, after which control returns to step 38. The other block of data need not be contiguous with the block just completed.

In the event the test at decision step 52 is positive, the F ratio is computed for the group of two comparison blocks at step 56. The results obtained in step 56 are submitted to peak detection in step 60. Peak detection optionally includes statistical tests of significance, as is explained hereinbelow.

If peaks are detected, then the boundary of a guard interval is established in step 62 for purposes of synchronization of the FFT window which is necessary for further signal reconstruction. If peaks are not detected, the above process is repeated with a block of samples taken from another portion of the data stream.

Example 1:

Referring now to Fig. 7 a complex signal was generated according to the above noted European Telecommunications standard using a random number generator, and transmitted across a Ricean channel model together with added white Gaussian noise (SNR = 3.7). Data symbols were then analyzed according to the above described method. The results 6 data symbols are shown in Fig. 7, wherein the F ratio is plotted for convenience of presentation on a logarithmic axis as line 64, because the spikes 66, 68, at the beginning and end of the guard intervals respectively, are very large.

Although it is quite evident from Figure 7 that the ends of the guard intervals are easy to find using any of several well known peak detectors, it is possible to apply a statistical test to more accurately answer the question: do the two blocks of samples have the same dispersion? This is the null hypothesis, H_0, i.e. the dispersion is the

same and the observed spike in F is due to random fluctuations only. If H_0 has very low probability it can be rejected, which would correspond to detection of the start or end of the guard interval. From the way the COFDM symbol is constructed H_0 is expected to be true for comparison blocks lying entirely within the guard interval or within the active interval, but false when the comparison blocks straddle a boundary at the start or end of the guard interval. If comparison blocks of random samples are drawn from the same population then the probability of F is given by

$$Q(F|v_1,v_2) = I_x(\frac{v_1}{2},\frac{v_2}{2}) \quad \textbf{(17)}$$

where I() is the incomplete Beta function,

$$x = \frac{v_2}{v_2 + v_1 F} \quad \textbf{(18)}$$

and v_1 and v_2 are the number of degrees of freedom with which the first and second dispersions are estimated. In this example v1 = v2 = (N-1) if n >= N. The shape of the function is shown in Fig. 8. From a statistical point of view n should be sufficiently large so that the two blocks do not overlap, i.e. n >= N. If the blocks do overlap, then the calculation of the second dispersion will use samples used for the calculation of the first dispersion. This effectively reduces the number of degrees of freedom and hence the significance of the result. It has been determined that setting n=N works well.

The function Q() in equation (13) actually gives the one-tailed probability. H_0 could be rejected if F is either very large or very small, and so the two-tailed test is required. Actually the two tails are identical, so for a two-tailed test the probability is double that given in equation (13). However, this results in values of probability greater than one for F<1. The probability, p, is therefore calculated as follows:

$$p = 2I_x(\frac{v_1}{2},\frac{v_2}{2}) \quad \textbf{(19)}$$

and then, if (p > 1), p = 2 - p. This probability reflects the viability of H_0. Thus if p is small, H_0 can be rejected and it can be stated, with a specified degree of certainty, that the comparison blocks come from sample populations with different dispersion. The noted European Telecommunications Standard specification states that the block size, N, should be 32 for a correlation algorithm. N={32,64} have been successfully tried. The probability functions obtained are shown in Fig. 9 using these values for N. In the preferred embodiment p <= 0.05 has been set for the rejection of H_0.

A precise implementation would be to calculate F, then x, then the incomplete Beta function, then p and then apply the threshold test. This algorithm would be very difficult to realize in hardware since the Beta function is very complicated. In the preferred embodiment it is much simpler, and gives the same results, to set the acceptance threshold and N parameter, and thus define an upper and lower limit for F. It is then only necessary to calculate F and compare it with the limits. In order to simply find the end of the guard interval it may be safely assumed that F>1. Only the upper limit on F is needed. To calculate the limits on F accurately, a suitable root-finding method, such as Newton-Raphson may be utilized. Typical values are given in Table 1.

Table 1

p threshold	v1 = v2 = 31		v1 = v2 = 63	
	F_lower	F_upper	F_lower	F_upper
0.2	0.627419	1.593832	0.722591	1.383909
0.1	0.548808	1.822132	0.658620	1.518326
0.05	0.488143	2.048582	0.607525	1.646022
0.01	0.386894	2.584689	0.518205	1.929738
0.005	0.354055	2.824422	0.487936	2.049448
0.001	0.293234	3.410251	0.429794	2.326695
10^{-4}		4.337235		
10^{-5}		5.393528		
10^{-6}		6.605896		
10^{-7}		8.002969		
10^{-8}		9.616664		

This method has been successfully tested using the specified channel model with additive white Gaussian noise (SNR=3.7).

The formula for dispersion given in Equation (12) would require a multiplier for implementation in silicon. The calculation of F is a division in which the (N-1) normalisation constants cancel out as long as the two blocks have the same size. Accurate multiplication and division can be expensive in silicon. In the preferred embodiment simplifications have been implemented which give less accurate, but still viable, values for F. S_i can be assumed to have zero mean so it is not necessary to calculate the mean from the block of samples. This also increases the number of degrees of freedom from (N-1) to N. Instead of calculating variance using the standard

sum of squares formula, the dispersion can be estimated by the mean absolute deviation. The formula for VAR(i) becomes

$$VAR(i) = \frac{1}{N}\left(\sum_{j=0}^{N-1} |s_{i-j}|\right)^2 \qquad (20)$$

The (1/N) factor divides out in the calculation of F if the two blocks have the same size. But there still remains the division of the two dispersions and the squaring required. These can be tackled using logarithms to the base 2. Substituting from Equation (16) into Equation (11) gives

$$F = \left(\frac{\sum_{j=0}^{N-1} |s_{i-j}|}{\sum_{j=0}^{N-1} |s_{i-n-j}|}\right)^2 = \left(\frac{S_a}{S_b}\right)^2 \qquad (21)$$

Taking logs to the base 2 gives

$$\log F = 2(\log s_a - \log s_b) = y \qquad (22)$$

It is then only necessary to calculate y and compare it with the logarithm to the base 2 of the F upper limit. The comparison can be made by subtracting the log of the limit from 2(log2sa-log2sb) and comparing with zero. The factor of 2 can be absorbed into the limit.

Calculation of the logs to base two is relatively straightforward in hardware if the numbers are stored as fixed point fractions. The fractions can be split into an exponent and a fractional mantissa: $x = A2^B$. Taking log base 2 gives logx = logA + B. Since A is fractional it is practical to find its logarithm using a lookup table. The exponent B can be found from the position of the MSB (since s_a and s_b will both be positive numbers).

The calculation can thus be reduced to require only addition and subtraction arithmetic operations. The limit should also be recalculated using v1=v2=N if using this method. In practice, the significance level may be set empirically for a particular application, preferably p = 0.05.

It will be appreciated by those skilled in the art that various measures of dispersion may be utilized without departing from the spirit of the invention, for example the

standard deviation, skew, various moments, histograms, and other calculations known in the art.

In a first alternate embodiment of the invention, the above described method is employed using either the real or the imaginary parts of the signal instead of the modulus. This embodiment achieves economy in hardware.

In a second alternate embodiment of the invention, the n parameter of equation (11) has been optimized. At the end of the guard interval, the two blocks straddle more of the transition to the active interval, giving a well-defined increase in the dispersion. Using any value n>2 has the drawback that several successive points will give significant increases as the later block travels up to the boundary. This small problem is easily overcome by introducing a dead period after detection of the boundary. That is, once a spike has been detected a set of samples equal to the size of the FFT window is accepted before further attempts are made to locate another spike. The dead period has the added benefit of not introducing false spikes. When using larger values of n the spikes 66, 68 (Fig. 7) increase, whilst the H_0 noisy F signal remain much the same.

Example 2:

The maximum F-spike height as a function of n has been measured systematically together with the background variation in F. The results are shown in Table 2.

Table 2

(1)	(2)	(3)	(4)	(5)
n	<F>	$F_{s.d.}$	F_{max}	(4) / (3)
3	1.0009	0.07	7.5	107
5	1.0012	0.10	10.7	107
10	1.0011	0.10	12.9	92
15	1.0014	0.17	16.7	98
20	1.0014	0.19	19.3	102
30	1.0012	0.23	20.9	91
40	0.9975	0.24	22.0	92
50	0.9926	0.25	20.4	81.6

Table 2 was developed using the first 5 frames of the signal analyzed in Fig. 7. The statistics in columns (2) and (3) of Table 2 were made by excluding any points where F>=3.0 to exclude spikes from the calculations. The spikes would otherwise affect the values of mean and standard deviation even though they are from a different statistical population.

The results indicate that the background variation in F, $F_{s.d.}$, was affected by n, increasing asymptotically to a value of approximately 0.28. It is likely that this is the effect of overlapping blocks. For example, for N=64 and n<64, the blocks over which the dispersions are calculated will contain some of the same values and therefore be correlated. To test this theory Fs.d. was evaluated for n>N, and the results are shown in Table 3.

Table 3

n	$F_{s.d.}$
60	0.258
70	0.266
80	0.270
90	0.278
100	0.278
128	0.297
256	0.366

The dependence becomes linear at n >= N/2. If F is calculated every n samples, rather than every sample, then this dependence may be reduced. However, this creates a risk for small guard intervals of not having the first block wholly within the guard interval and the second wholly within the active interval.

A third alternate embodiment of the invention is disclosed with reference to Fig. 10, which schematically illustrates a timing synchronization circuit 70. The circuit accepts a complex input signal 72, and includes a circuit module 74 which develops the modulus of its input, which is taken from node 83. The circuit module 74 insures that the value being subsequently processed is an unsigned number. The input to the circuit module 74 is a difference signal which is developed by a subtracter 75 which takes as inputs the input signal 72 and a delayed version of the input signal 72 which has been processed through a delay circuit 79, preferably realized as a FIFO 77 of length L, where L is the size of the FFT window. As explained above, it is also possible to operate this circuit where the input signal 72 is real, imaginary, or complex, or even the modulus of a complex number. In the case where the input signal 72 is real, or imaginary, the circuit module 74 can be modified, and can be any known circuit that removes the sign of the output of the subtracter 75, or equivalently sets the sign so that the outputs accumulate monotonically; i.e. the circuit has a unipolar output. The output of the circuit module 74 is ultimately clocked into a digital delay, which is preferably implemented as a FIFO 78. When the FIFO 78 is full, a signal SIG1 80 is asserted, and the output of the FIFO 78

becomes available, as indicated by the AND gate 82. An adder/subtracter circuit 84 is also connected to the node 76, and its output is stored in a register 86. A delayed version of the output of the adder/subtracter circuit 84 is taken from the register 86 and fed back as a second input to the adder/subtracter circuit 84 on line 88. In the event the signal SIG1 80 has been asserted, a version of the output of the circuit module 74, delayed by a first predetermined interval N, where N is the number of samples in the comparison blocks, is subtracted from the signal on node 76.

The signal on line 88 is an index into a lookup table, preferably implemented as a read-only-memory ("ROM"), and shown as ROM 90. The address of the ROM 90 contains the logarithm to the base 2 of the magnitude of the signal on line 88, which then appears at node 92. The node 92 is connected to a subtracter 94, and to a delay circuit, shown as FIFO 98, which is used to develop the denominator of the middle term of equation (17).

The subtracter 94 produces a signal which is compared against the $\log_2$ of a predetermined threshold value F_{LIMIT} in a comparison circuit 106, shown for simplicity as an adder 108 connected to a comparator 110. The output signal SYNC 112 is asserted when the boundary of a guard interval has been located.

Although not implemented in the presently preferred embodiment, It is also possible to configure the size of the FIFO 77 dynamically, so that the size of the interval being evaluated can be adjusted according to operating conditions. This may conveniently be done by storing the values on the node 92 in a RAM 114 for computation of their dispersion.

In a fourth alternate embodiment of the invention, explained with reference to Fig. 11, components similar to those of the embodiment shown in Fig. 10 have the same reference numerals. A timing synchronization circuit 116 is similar to the timing synchronization circuit 70, except now the delay circuit 79 is realized as the FIFO 77, and another FIFO 100, one of which is selected by a multiplexer 102. Both of the FIFOs 77, 100 provide the same delay; however the capacities of the two are different. The FIFO 100 provides for storage of samples taken in an interval equal to the size of the FFT window, and is normally selected in a first mode of operation, for example during channel acquisition, when it is necessary to evaluate an entire symbol in order to locate a boundary of a guard interval. In the noted European Telecommunications standard, up to 8K of data storage is required, with commensurate resource requirements. During subsequent operation, the approximate location of the guard interval boundaries will be known from the history of the previous symbols. In a second mode of operation, It is therefore only necessary to evaluate a much smaller interval in order to verify the exact location of the guard interval boundary. The number of samples used in the computation

of the dispersion can be kept to a small number, preferably 32 or 64, and the much smaller FIFO 77 accordingly selected to hold the computed values. The resources saved thereby can be utilized for other functions in the demodulator, and memory utilized by the larger FIFO 100 may also be reallocated for other purposes.

A control block 81 optionally advances the evaluation interval relative to symbol boundaries in the data stream in successive symbols, and can also be used to delay for the dead period. Eventually the moving evaluation interval straddles the boundary of the current symbol's guard interval, and synchronization is then determined. The size of the evaluation interval is chosen to minimize the use of memory, yet to be large enough to achieve statistical significance in the evaluation interval. The size of the evaluation interval, and the FIFO 77 may be statically or dynamically configured.

Single Chip Implementation of a COFDM Demodulator

Overview

Referring initially to Fig. 12, there is shown a high level block diagram of a multicarrier digital receiver 126 in accordance with the invention. The embodiment described hereinbelow conforms to the ETS 300 744 telecommunications standard (2K mode), but can be adapted by those skilled in the art to operate with other standards without departing from the spirit of the invention. A radio frequency signal is received from a channel such as an antenna 128, into a tuner 130, which is conventional, and preferably has first and second intermediate frequency amplifiers. The output of the second intermediate frequency amplifier (not shown), is conducted on line 132 to an analog to digital converter 134. The digitized output of the analog to digital converter 134 is provided to block 136 in which I/Q demodulation, FFT, channel estimation and correction, inner and outer deinterleaving, and forward error correction are conducted. Carrier and timing recovery are performed in block 136 entirely in the digital domain, and the only feedback to the tuner 130 is the automatic gain control ("AGC") signal which is provided on line 138. A steady 20 MHz clock on line 140 is provided for use as a sampling clock for the external analog to digital converter 134. A host microprocessor interface 142 can be either parallel or serial. The system has been arranged to operate with a minimum of host processor support. In particular channel acquisition can be achieved without any host processor intervention.

The functions performed within the block 136 are grouped for convenience of presentation into a front end (Fig. 13), FFT and channel correction group (Fig. 14), and a back end (Fig. 15).

As shown in Fig. 13, I/Q samples at are received by an IQ demodulator 144 from the analog to digital converter 134 (Fig. 12) on a bus 146 at a rate of 20 megasamples per second. An AGC circuit 148 also takes its input from the bus 146. A frequency rate

control loop is implemented using a numerically controlled oscillator 150, which receives frequency error signals on line 152, and frequency error update information on line 154. Frequency and sampling rate control are achieved in the frequency domain, based on the pilot carrier information. The frequency error signals, which are derived from the pilot carriers, and the frequency error update information will both be disclosed in further detail shortly. The I and Q data output from the IQ demodulator 144 are both passed through identical low pass filters 156, decimated to 10 megasamples per second, and provided to a sinc interpolator 158. Sample rate control is achieved using a numerically controlled oscillator 160 which receives sample rate control information derived from the pilot signals on line 162, and receives sample error update timing information on line 164.

As shown in Fig. 14, acquisition and control of the FFT window are performed in block 166, which receives signals from the sinc interpolator 158 (Fig. 13). The FFT computations are performed in FFT calculation circuitry 168. Channel estimation and correction are performed in channel estimation and correction block 170, and involves localization of the pilot carriers, as will be described below in greater detail. The tps information obtained during pilot localization is processed in tps sequence extract block 172. Uncorrected pilot carriers are provided by the circuitry of channel estimation and correction block 170 to correction circuitry 174, which develops sampling rate error and frequency error signals that are fed back to the numerically controlled oscillators 150, 160 (Fig. 13).

Referring to Fig. 15, corrected I and Q data output from channel estimation and correction block 170 are provided to demapping circuitry 176. The current constellation and hierarchical constellation parameters, derived from the tps data, are also input on lines 178, 180. The resulting symbols are deinterleaved in symbol deinterleaver 182, utilizing a 1512 x 13 memory store. One bit of each cell in the memory store is used to flag carriers having insufficient signal strength for reliable channel correction. Bit deinterleaver 184 then provides deinterleaved I and Q data to a Viterbi Decoder 186, which discards the flagged carriers, so that unreliable carriers do not influence traceback metrics. A Forney deinterleaver 188 accepts the output of the Viterbi Decoder 186 and is coupled to a Reed-Solomon decoder 190. The forward error correction provided by the Viterbi and Reed-Solomon decoders is relied upon to recover lost data in the case of flagged carriers.

Referring to Fig. 16, in the presently preferred embodiment a mean value is calculated in block 192 for uncorrected carriers with reference to the previous symbol. Data carriers whose interpolated channel response falls below some fraction, preferably 0.2, of this mean will be marked with a bad_carrier flag 194. The bad_carrier flag 194

is carried through the demapping circuitry 176, symbol deinterleaver 182, and bit deinterleaver 184, to the Viterbi Decoder 186 where it is used to discard data relating to the unreliable carriers. The parameters used to set the bad_carrier flag 194 can be varied by the microprocessor interface 142.

An output interface 196 produces an output which can be an MPEG-2 transport stream. The symbol deinterleaver 182, and the bit deinterleaver 184 are conventional. The Viterbi decoder 186, Forney deinterleaver 188, Reed-Solomon decoder 190, and the output interface 196 are conventional. They can be the components disclosed in copending Application No. 638,273, entitled "An Error Detection and Correction System for a Stream of Encoded Data", filed April 26, 1996, Application No. 480,976, entitled "Signal Processing System", filed June 7, 1995, and Application No. 481,107, entitled "Signal Processing Apparatus and Method", filed June 7, 1995, all of which are commonly assigned herewith, and are incorporated herein by reference. The operation of the multicarrier digital receiver 126 (Fig. 12) is controlled by a system controller 198.

Optionally the hierarchical constellation parameters can be programmed to speed up channel acquisition, rather than derived from the tps data.

The input and output signals and the register map of the multicarrier digital receiver 126 are described in tables 4, and 5 respectively.

Automatic Gain Control

The purpose of the AGC circuit 148 (Fig. 13)is to generate a control signal to vary the gain of the COFDM input signal to the device before it is analog-to-digital converted. As shown in greater detail in Fig. 17, a Sigma-Delta modulator 200 is used to provide a signal which can be used as a gain control to a tuner, once it has been low-pass filtered by an external R-C network.

The magnitude of the control voltage signal 202 is given by:

$$control\_voltage = control\_voltage - error \quad \textbf{(23)}$$

where

$$error = K(|data| - mean) \quad \textbf{(24)}$$

where K is a constant (normally K<<1) which determines the gain in the AGC control loop. The mean value can be determined from the statistics of Gaussian noise, which is a close approximation to the properties of the COFDM input signal, where the input data is scaled to +/-1. The control voltage signal 202 is set back to its initial value when the signal resync 204 is set low, indicating a channel change or some other event requiring resynchronization.

The input and output signals and the registers for the microprocessor interface 142 of the AGC circuit 148 are described in tables 6, 7, and 8 respectively.

IQ Demodulator

The function of the IQ demodulator 144 (Fig. 13) is to recover in-phase and quadrature components of the received sampled data. It is shown in further detail in Fig. 18.

The numerically controlled oscillator 150 generates in-phase and quadrature sinusoids at a rate of (32/7) MHz, which are multiplied with data samples in multipliers 206. The address generator 208 advances the phase linearly. The frequency error input 210 increments or decrements the phase advance value. The samples are multiplied with the sinusoids in the multipliers 206using 10 bit x 10 bit multiply operations. In one embodiment the IQ demodulator 144 is operated at 20 MHZ and then retimed to 40MHz in retiming block 212. In a preferred embodiment the IQ demodulator 144 is operated at 40MHz, in which case the retiming block 212 is omitted.

Sinusoids are generated by the address generator 208 on lines 214, 216. The phase value is employed as an address into a lookup table ROM 218. Only quarter cycles are stored in the lookup table ROM 218 to save area. Full cycles can be generated from the stored quarter cycles by manipulating the data from the ROM 218 and inverting the data in the case of negative cycles. Two values are read from the lookup table ROM 218 for every input sample -- a cosine and a sine, which differ in phase by 90 degrees.

The input and output signals of the IQ demodulator 144 are described in tables 9 and 10 respectively.

Low Pass Filter

The purpose of the low pass filters 156 (Fig. 13) is to remove aliased frequencies after IQ demodulation - frequencies above the 32/7 MHz second IF are suppressed by 40dB. I and Q data are filtered separately. The output data is decimated to 10 megasamples per second ("Msps") because the filter removes any frequencies above 1/4 of the original 20 Msps sampling rate. The filter is constructed with approximately 60 taps which are symmetrical about the center, allowing the filter structure to be optimized to reduce the number of multipliers 220. Fig. 19 is a block diagram of one of the low pass filters 156, the other being identical. Fig. 19 shows a representative symmetrical tap 222, and a center tap 224. The required filter response of the low pass filters 156 is shown in Fig. 20.

The input and output signals of the low pass filters 156 are described in tables 11 and 12 respectively.

Resampling

Referring to Fig. 13, the purpose of resampling is to reduce the 10 Msps data stream output from the low pass filters 156 down to a rate of (64/7) Msps, which is the

nominal sample rate of the terrestrial digital video broadcasting ("DVB-T") modulator at the transmitter.

Resampling is accomplished in the sinc interpolator 158, and the numerically controlled oscillator 160. The latter generates a nominal 64/7 MHZ signal. The resampling circuitry is shown in further detail in Fig. 21. The numerically controlled oscillator 160 generates a valid pulse on line 226 and a signal 228 representing the interpolation distance for each 40MHz clock cycle in which a 64/7MHz sample should be produced. The interpolation distance is used to select the appropriate set of interpolating filter coefficients which are stored in coefficient ROMs 230. It should be noted that only the sinc interpolator for I data is illustrated in Fig. 21. The structures for Q data are identical.

Fig. 22 illustrates the generation of the interpolation distance and the valid pulse. Nominally T_s = 1/10 Msps, and T = 1/ (64/7) Msps. The sinc interpolation circuit disclosed in our noted Application No. 08/638,273 is suitable, with appropriate adjustment of the operating frequencies.

The input and output signals of the sinc interpolator 158 and the numerically controlled oscillator 160 are described in tables 13 and 14 respectively.

FFT Window

As has been explained in detail above, the function of the FFT Window function is to locate the "active interval" of the COFDM symbol, as distinct from the "guard interval". This function is referred to herein for convenience as "FFT Window". In this embodiment the active interval contains the time domain representation of the 2048 carriers which will be recovered by the FFT itself.

The FFT window operates in two modes; Acquisition and Tracking. In Acquisition mode the entire incoming sample stream is searched for the guard interval/active interval boundary. This is indicated when the F-ratio reaches a peak, as discussed above. Once this boundary has been located, window timing is triggered and the incoming sample stream is searched again for the next guard interval/active interval boundary. When this has been located the length of the guard interval is known and the expected position of the next guard/active boundary can be predicted. The FFT window function then switches to tracking mode.

This embodiment is similar to the fourth alternate embodiment discussed above in respect of the tracking mode. In tracking mode only a small section of the incoming sample stream around the point where the guard/active boundary is expected to be is searched. The position of the active interval drifts slightly in response to IF frequency and sampling rate offsets in the front-end before the FFT is calculated. This drift is

tracked and FFT window timing corrected, the corrections being inserted only during the guard interval.

It will be appreciated by those skilled in the art that in a practical single chip implementation as is disclosed herein, memory is an expensive resource in terms of chip area, and therefore must be minimized. Referring to Fig. 23, during Acquisition mode the FFT calculation process is not active so hardware can be shared between the FFT Window and the FFT calculation, most notably a 1024x22 RAM 232 used as a FIFO by the FFT Window, and selected for receipt of FFT data on line 234 by a multiplexer 236. Once in Tracking mode the FFT calculation process is active so that other control loops to recover sampling rate and frequency which depend on FFT data (e.g. pilots in the COFDM symbol) can initialize. Therefore tracking mode requires a dedicated tracking FIFO 238, which is selected by a multiplexer 240.

The input and output signals, and signals relating to the microprocessor interface 142 of the FFT Window circuitry shown in Fig. 23 are described in tables 15, 16, and 17 respectively.

In one embodiment a threshold level, set from statistical considerations, is applied to the F-ratio signal (see Fig. 7) to detect the negative and positive spikes which occur at the start and end of the guard interval respectively. The distance between the spikes is used to estimate the guard interval size. Repeated detection of the positive spikes is used to confirm correct synchronization. However with this method under noisy conditions the F-ratio signal becomes noisy and the spikes are not always reliably detectable.

In another embodiment peak detection is used to find the spikes in the F-ratios. It has been found that a fixed threshold is reliable only at or exceeding about a carrier-to-noise ("C/N") ratio of 12 dB. Peak detection is generally more sensitive and more specific, with generally reliable operation generally at 6 - 7 dB. The maxima should occur at the end of the guard interval. The difference in time between the two maxima is checked against the possible guard interval sizes. With an allowance for noise, the difference in time indicates the most likely guard interval size and the maxima themselves provide a good indication of the start of the active part of the symbol.

Preferably this process is iterated for several symbols to confirm detection, and is expected to improve performance when the C/N ratio is low.

The data stream is passed to accumulators 242, 244, each holding 64 moduli. Conversion to logarithms and subtraction of the logarithms is performed in block 246. The peaks are detected in peak detector block 248. Averaging of the symbol peaks is performed in block 250.

In noisy conditions, the maxima may be due to noise giving possibly inaccurate indications of the guard interval length and the start of the active symbol. The general strategy to cope with this is to perform a limited number of retries.

Currently, calculation of the F-ratio is done "on the fly" i.e. only once at each point. The variance estimates are calculated from 64 values only. Under noisy conditions, the variance estimates become very noisy and the spikes can become obscured. In an optional variation this problem is solved by obtaining more values for the variance estimate, by storing the variance estimate during acquisition for each of the possible $T+G_{max}$ points in the storage block 256. The variance estimates themselves may be formed by accumulating variances for each point, and then filtering in time over a number of symbols. A moving average filter or an infinite impulse response ("IIR") filter is suitable A moving run of symbols, preferably between 16 and 32, are integrated in block 252 which increases the reliability of peak detection under noisy conditions. The storage block 256 holding the integrated F-ratio values is searched to find the maximum value This is of length $T+G_{max}$, where G_{max} is the maximum guard interval size, T/4. Preferably the memory for storage block 256 is dynamically allocated, depending on whether acquisition mode or tracking mode is operative. Any unused memory is released to other processes. Similarly in tracking mode the integrated data stream is stored in tracking integration buffer 254.

This method has been tested with up to 4 symbols, without an IIR filter, and it has been found that the spikes can be recovered. However this approach does require increased memory.

FFT Processor

The discrete Fourier transform ("DFT") has the well known formula

$$x(k) = \frac{1}{L}\sum_{n=0}^{L-1} x(n)W^{nk} \qquad k = 0,1,\ldots,N-1 \qquad \textbf{(25)}$$

where N = the number of points in the DFT;

x(k) = the kth output in the frequency domain;

x(n) = the nth input in the time domain

and

$$W_L^{nk} = e^{-j(2\pi nk/L)} \qquad \textbf{(26)}$$

W is also known as a "twiddle factor".

For N > 1000 the DFT imposes a heavy computational burden and becomes impractical. Instead the continuous Fourier transform is used, given by

$$x(t) = \int_{t=-\infty}^{t=+\infty} x(t)e^{-j\omega t}dt \qquad (27)$$

The continuous Fourier transform, when computed according to the well known FFT algorithm, breaks the original N-point sequence into two shorter sequences. In the present invention the FFT is implemented using the basic butterfly unit 258 as shown in Fig. 24. The outputs C and D represent equations of the form C = A + B, and D = (A - B)W^k. The butterfly unit 258 exploits the fact that the powers of W are really just complex additions or subtractions.

A real-time FFT processor, realized as the FFT calculation circuitry 168 (Fig. 14) is a key component in the implementation of the multicarrier digital receiver 126 (Fig. 12). Known 8K pipeline FFT chips have been implemented with 1.5M transistors, requiring an area of 100 mm^2 in 0.5μ technology, based on the architecture of Bi and Jones. Even using a memory implementation with 3-transistor digital delay line techniques, over 1M transistors are needed. This has been further reduced with alternative architecture to 0.6M, as reported in the document *A New Approach to Pipeline FFT Processor*. Shousheng He and Mats Torkelson, Teracom Svensk RundRadio. DTTV-SA 180, TM 1547. This document proposes a hardware-oriented radix-2^2 algorithm. having radix-4 multiplicative complexity. However the requirements of the FFT computation in the present invention require the implementation of a radix 2^2+2 FFT processor.

Referring to Fig. 25 and Fig. 26 the butterfly structures BF2I 260 and BF2II 262, known from the noted Torkelson publication, are shown. The butterfly structure BF2II 262 differs from the butterfly structure BF2I 260 in that it has logic 264 and has a crossover 266 for crossing the real and imaginary inputs to facilitate multiplication by -j.

Fig. 27 illustrates the retimed architecture of a radix 2^2 + 2 FFT processor 268 in accordance with the invention, which is fully pipelined, and comprises a plurality of stages. stage-0 270 through stage-6 272. Except for stage-0 270, the stages each comprise one butterfly structure BF2I 260 and one butterfly structure BF2II 262, and storage RAMS 274, 276 associated therewith. stage-0 270 only has a single butterfly structure BF2I 260. This architecture performs a straight-forward 32-point FFT. stage-6 272 has control logic associated therewith, including demultiplexer 278 and multiplexer 280, allowing stage-6 272 to be bypassed, thus providing a 2K implementation of the FFT. Counters 282 configure the butterfly structures BF2I 260 and BF2II 262 to select one of the two possible diagonal computations, during which data is being simultaneously written to and read from the storage RAMS 274, 276.

Fig. 28 illustrates a 32 point flow graph of the FFT processor 268 using radix 2^2+2 pipeline architecture. Computations are performed using eight 4-point FFTs and four 8-point FFTs. These are decomposed in turn into two 4-point FFTs and four 2-point FFTs.

Fig. 29 illustrates the retimed architecture of a configurable 2K/8K radix 2^2+2 single path, delay feedback pipelined FFT processor 284, in which like elements in Fig. 27 are given the same reference numerals. The stages have a plurality of pipeline registers 286 which are required for proper timing of the butterfly structures BF2I 260 and BF2II 262 in the various stages. As can be seen, the addition of each pipelined stage multiplies the range of the FFT by a factor of 4. There are 6 complex multipliers 288, 290, 292, 294, 296, 298 which operate in parallel. This processor computes one pair of I/Q data points every four fast clock cycles, which is equivalent to the sample rate clock. Using 0.35µm technology the worst case throughput is 140µs for the 2K mode of operation, and 550µs for the 8K mode, exceeding the requirements of the ETS 300 744 telecommunications standard. Data enters the pipeline from the left side of Fig. 29, and emerges on the right. The intermediate storage requirements are 2K/8K for I data and 2K/8K for Q data, and is mode dependent. In practice the radix-4 stage is implemented as a cascade of two adapted radix-2 stages that exploit the radix-4 algorithms to reduce the number of required complex multipliers.

Fig. 30 is a schematic of one embodiment of the multipliers 288, 290, 292, 294, 296, 298 for performing the complex multiplication C = A x B, where A is data, and B is a coefficient. Because the FFT processor 284 has 6 complex multipliers, each requiring 3 hardware multipliers 300, a total of 18 hardware multipliers 300 would be required. It is preferable to use the embodiment of Fig. 31 in which some of the hardware multipliers 300 are replaced by multiplexers 302, 304.

Turning again to Fig. 29 there are a plurality of RAMS 306, 308, 310, 312, 314, 316 which are preferably realized as ROMs and contain lookup tables containing complex coefficients comprising cosines for the multipliers 288, 290, 292, 294, 296, 298 respectively. It has been discovered that by addressing the RAMS 306, 308, 310, 312, 314, 316 according to a particular addressing scheme, the size of these RAMS can be markedly reduced. The tradeoff between the complexity of the addressing circuitry and the reduction in RAM size becomes favorable beginning at stage-3 318. Referring again to Fig. 28 there are two columns 320, 322. Column 320 holds values W^2 - W^{14}, followed by W^1 - W^7, and then W^3 - W^{21}. These coefficients are stored in the RAM 308, required by the particular multiplier 290. Column 322 contains values W^8, W^4, W^{12}, which repeat 3 times. Note further that between the values W^8, W^4, and W^4, W^{12} are connections 324, 326 to the preceding butterfly unit located in column 328. In practice the connections 324, 326 are implemented as multiplications by W^0. In moving from multiplier to

multiplier toward the left in Fig. 29, the lookup table space is multiplied by a power of 4 at each stage. In Fig. 32 table 330, the lookup table for multiplier M^3 contains 512 entries. It can be deduced by extrapolation that multiplier M^5 must contain 8192 twiddle factors, and corresponds to the size of the FFT being performed by the FFT processor 284 (Fig. 29).

Before examining the look-up table space in more detail it is helpful to consider the plurality of horizontal lines 332. Moving downward from the top of Fig. 28, the line beginning at x(3) extends to W^8, which is the first twiddle factor required, and is at the third effective step in the flow diagram. Figs. 33 and 32 show the organization of the twiddle factors for each of the multipliers, wherein the terminology M_k represents the multiplier associated with the kth stage. Thus table 334 relates to multiplier M_0. The notation for the W values (twiddle factors) is shown in box 336. The subscript "B" at the bottom right represents a time stamp, that is an order dependency in which the twiddle factors are required by the pipeline. The superscript "A" represents the address of the twiddle factor in its lookup table. The superscript "N" is the index of the twiddle factor.

Thus in table 334 it may be seen that W^0 is required at time 0, W^1 at time 1, and W^0 is again required at time 2. Further inspection of the other tables in Figs. 33, 32 reveals that half of the entries in each table are redundant. The storage requirement for the lookup tables can be decreased by 50% by eliminating redundant entries. This has been accomplished by organizing the W values in ascending order by index, so that the values can be stored in memory in ascending order. Thus in the case of table 338 the index values range from 0 to 21, with gaps at 11, 13, 16, 17, 19, and 20.

The procedure for organizing the lookup table and the addressing scheme for accessing the twiddle factors is explained with reference to table 338, but is applicable to the other tables in Fig. 33. (1) Each row is assigned a line number as illustrated. (2) Each twiddle factor is assigned an order dependency which is noted in the lower right of its respective cell in table 338. (3) It is assumed that table 338 in its reduced form will contain only unique twiddle factors in ascending order by index within the memory address space. Consequently each twiddle factor is assigned a memory address as shown in the upper left of its respective cell.

During address generation, for line 3 of table 338 the address is simply held at 0. For line 1 the address is incremented by 1 to the end of the line. However lines 0 and 2 contain non-trivial address sequences. For line 0, looking at table 340, which contains 64 values, it will be observed that the address sequence changes according to the intervals 2,2,2,2, and then later 1,1,2,1,1,2 ... For line 2, the address first increments by 3, then by 2, and finally by 1. The locations at which the address increments change are

referred to herein as the "break-points". These values of the break points range between 0, corresponding to the first point in line 2, to the last position in the line.

By inspection it can be seen that the occurrence of the first break point changes from table to table following the recurrence relationship

$$B1_{M_N} = 4B1_{M_{N-1}} \quad (28)$$

with the initial condition

$$B1_{M_0} = 1 \quad (29)$$

where M_N is the multiplier of the Nth stage of the FFT processor 284.

Expanding the recurrence relationship gives:

$$B1_{M_N} = (((4B1_{M_0} - 1)x4 - 1)x4 - 1) \ldots \quad (30)$$

$$B1_{M_N} = 4^N B1_{M_0} - 4^{N-3} - 4^{N-2} \ldots - 4^0 \quad (31)$$

$$B1_{M_N} = 4^N B1_{M_N} - \sum_{n=0}^{N-1} 4^n \quad (32)$$

Similarly the second break point B2 for line 2 is determined from the recurrence relation

$$B2_{M_N} = 4B2_{M_{N-1}} + 1 \quad (33)$$

with the initial condition

$$B2_{M_0} = 1 \quad (34)$$

or

$$B2_{M_N} = (((4B2_{M_0} + 1)x4 + 1)x4 + 1) \ldots \quad (35)$$

$$B2_{M_N} = \sum_{n=0}^{N} 4^n \quad (36)$$

Break point B3 for line 0 at which the sequence changes from increments of 2,2,2,2 to the pattern 1,1,2,1,1,2 . . . can be located by inspecting tables 338, 340, and 330. In table 338 the break point B3 occurs very late in the line, such that the second sequence only presents its first two elements. By examining the address locations in the larger noted tables, it can be deduced that the location of break point B3 is related to the number of entries in a particular table as

$$B3 = \frac{K}{4} + 2 \qquad (37)$$

where K is the number of table entries. In the tables in Fig. 29 K = 8, 32, 128, 2048, 8192. Therefore, in terms of the N'th complex multiplier, break point B3 can be expressed as

$$B3_{M_N} = 2 \times 4^N + 2 \qquad (38)$$

where $N \geq 0$.

Address generators 342, 344, 346, 348 are operative for the lookup tables in RAMS 310, 312, 314, 316. Silicon area savings for the smaller tables 308, 306 are too small to make this scheme worthwhile.

Fig. 34 schematically illustrates an address generator 342 for the above described address generation scheme, and is specific for the table 340 and multiplier M_2. 128 possible input states are accepted in lines in_Addr 350, and a multiplexer 352 selects the two most significant bits to decode 1 of 4 values. The output of the multiplexer 352 relates to the line number of the input state. Actually the output is the address increment applicable to the line number of the input state, and is used to control a counter 354 whose incremental address changes according to value on line 356. Thus, the increment for line 3 of table 340 is provided to the multiplexer 352 on line 358, and has a value of zero, as was explained above. Similarly the increment for line 1 of table 340 is provided to the multiplexer 352 on line 360, and has a value of 1.

The situations of line 0 and line 2 are more complicated. For line 0 the output of decoding logic 362 is provided by multiplexer 364, and has either an incremental value of 2, or the output of multiplexer 366. The latter could be either 1 or 2, depending on the state of a two bit counter 368, which feeds back a value of 0 or 1 as signal count 370.

Decoding logic 372 decodes the states for line 2 of table 340. The relationship of the current input state to the two break points of line 2 are tested by comparators 374, 376. The break point is actually set one sample earlier than the comparator output to allow for retiming. The outputs of the comparators 374, 376 are selectors for the multiplexers 378, 380 respectively.

The current address, held in accumulator 382 is incremented by the output of the multiplexer 352 by the adder 384. A simple logic circuit 386 resets the outgoing address, which is contained in register ACC 388, by asserting the signal rst 390 upon completion of each line of table 340. This insures that at the start of the next line the address points to twiddle factor W^0. The new address is output on the 6 bit bus out_Address 392, which is one bit smaller than the input in_Addr 350.

Fig. 35 is a generalization of address generator 342 (Fig. 34), in which the incoming address has a path of B bits. Like elements in Figs. 34 and 35 are given the same reference numerals. The structure of address generator 394 is similar to that of the address generator 342, except now the various lines of the input in_addr 396 and the output out_addr[B-2:0] 398 are denoted in terms of B. Thus the multiplexer 352 in Fig. 35 is selected by input in_addr [B-1:B-2] 400 . Similarly one of the inputs of comparator 374 and of comparator 376 is in_addr [B-3:0] 402. Out_addr[B-2:0] 398 forms the output. The advantage of this structure is a reduction in the size of the lookup table RAM of 50%.

The FFT calculation circuitry 168 (Fig. 14) is disclosed in Verilog code listings 1 - 17. The Verilog code for the address generator 394 is generic, enabling any power-of-four table to be implemented.

Channel Estimation and Correction

The function of the Channel estimation and correction circuitry shown in channel estimation and correction block 170 (Fig. 14) is to estimate the frequency response of the channel based on the received values of the continuous and scattered pilots specified in the ETS 300 744 telecommunications standard, and generate compensation coefficients which correct for the channel effects and thus reconstruct the transmitted spectrum. A more detailed block diagram of the channel estimation and correction block 170 is shown in Fig. 16.

In acquisition mode, the channel estimation and correction block 170 needs to locate the pilots before any channel estimation can take place. The circuitry performs a convolution across the 2048 carriers to locate the positions of the scattered pilots, which are always evenly spaced, 12 carriers apart. Having found the scattered pilots, the continual pilots can be located; once this is done the exact position of the 1705 active carriers within the 2048 outputs of the FFT calculation circuitry 168 (Fig. 14) is known. A timing generator 404 within the block can then be initialized, which then generates reference timing pulses to locate pilots for channel estimation calculation and for use in other functions of the demodulator as well.

Channel estimation is performed by using the evenly spaced scattered pilots, and then interpolating between them to generate the frequency response of the channel. The received carriers (pilots and data) are complex divided by the interpolated channel response to produced a corrected spectrum. A complete symbol is held in a buffer 406. This corrects for the bit-reversed order of the data received from the FFT calculation circuitry 168. It should be noted that raw, uncorrected data is required by the frequency and sampling rate error circuitry.

The task of synchronizing to the OFDM symbol in the frequency domain data received from the FFT calculation circuitry 168 (Fig. 14) begins with the localization of the scattered and continual pilots, which occurs in pilot locate block 408. Scattered pilots, which according to the ETS 300 744 telecommunications standard, occur every 12 data samples, offset by 3 samples with respect to the start of the frame in each succeeding frame. As the power of the pilot carriers is 4/3 the maximum power of any data carrier, a succession of correlations are performed using sets of carriers spaced at intervals of 12. One of the 12 possible sets is correlates highly with the boosted pilot carrier power.

A first embodiment of the pilot search procedure is now disclosed with reference to Figs. 36 and 16. It should be noted that the scattered pilot search procedure is done on the fly, and storage is only required in so far as is necessary to perform the subsequent step of continual pilot location discussed below. At step 410, after the assertion of the signal resync 204, generally occurring after a channel change or on power up, the signal pilot_lock 412 is set low. Then, at step 414 the process awaits the first symbol pulse from the FFT calculation circuitry 168 (Fig. 14) on line 416 indicating the start of the first symbol. The first symbol is received and stored. In one embodiment of the pilot search procedure each point from 0 to 2047 is read in turn, accumulating each value ($|I| + |Q|$) in one of 12 accumulators (not shown). The accumulators are selected in turn in a cycle of 12, thus convolving possible scattered pilot positions. Two well known peak trackers indicate the accumulator with highest value (Peak1) and the accumulator having the second highest value (Peak2). The accumulator having the highest value corresponds to the scattered pilot orientation. The second highest value is tracked so that the difference between the highest peak and the second highest peak can be used as a "quality" measure. At decision step 418, if the two peaks are not far enough apart, a test for completion of a full range frequency sweep is made at decision step 420. If the test fails, failure of the scattered pilot search is reported at step 422. Otherwise, at step 424 the IQ Demodulator LO frequency is incremented by +1/8 carrier spacing by incrementing the magnitude of the control signal freq_sweep 426. Then the search for scattered pilots is repeated after delaying 3 symbols at step 428 to allow time for the effect of the change to propagate through the FFT calculation circuitry 168 and buffers. The peak difference threshold can be altered by the control microprocessor via the microprocessor interface 142 and block 430.

In a variation of the first embodiment there is only a single peak tracker which indicates the accumulator with highest value, which corresponds to the scattered pilot orientation. The true scattered pilot orientation thus found is one of 12 possible orientations.

If the test at decision step 418 is successful, the search for continual pilots is begun at step 432 by establishing an initial pilot offset from the 0 location in the RAM, storing the FFT data, according to the formula

$$\text{pilot offset} = (\text{accumulator \# mod } 3) \quad \textbf{(39)}$$

Thus, if the scattered pilot peak is in accumulator 0, 3, 6 or 9 the pilot offset is 0. If the scattered pilot peak is in accumulator 1, 4, 7, or 10 then pilot offset is 1, etc. Then 45 carrier positions expected for continual pilots are read, adding the pilot offset value to the address, and accumulating ($|I| + |q|$) values. This procedure is repeated until first 115 continual pilot start positions have been searched. From the ETS 300 744 telecommunications standard the number of possible first carrier positions among the active carriers lying in a contiguous block between carrier 0 and carrier 2047 is easily calculated as (2048-1705) / 3 ≈ 115, as explained below. It is thus guaranteed that the active interval begins within the first (2048-1705) carrier positions. The carrier corresponding to the peak value stored is the first active carrier in the symbol.

Upon completion of the continual pilot search, at step 434 the timing generator 404 is reset to synchronize to the first active carrier and scattered pilot phase. The signal pilot_lock 412 is then set high at step 436, indicating that the pilots have been located successfully, then at step 436 the timing generator 404 is reset to synchronize to the first active carrier and scattered pilot phase.

In a tracking mode of operation, shown as step 438, the scattered pilot search is repeated periodically, and evaluated at decision step 440. This can be done at each symbol, or less frequently, depending upon propagation conditions. The predicted movement of the scattered pilot correlation peak is reflected by appropriate timing in the timing generator 404, and can be used as a test that timing has remained synchronized. Failure of the test at decision step 440 is reported at step 442, and the signal pilot_lock 412 is set low.

A second embodiment of the pilot search procedure is now disclosed with reference to Figs. 16 and 37. At step 444 the assertion of the signal resync 204, generally occurring after a channel change or on power up, the signal pilot_lock 412 is set low. Then, at step 446 a symbol is accepted for evaluation. A search for scattered pilots, conducted according to any of the procedures explained above, is performed at step 448. Then a search for continual pilots is performed as described above at step 450. At decision step 452 it is determined whether two symbols have been processed. If the test fails, control returns to step 446 and another symbol is processed. If the test succeeds at step 454 another test is made for consistency in the positions of the scattered and continual pilots in the two symbols. If the test at step 454 fails, then the

procedure beginning with decision step 420 is performed in the same manner as previously described with reference to Fig. 36. If the test at step 454 succeeds at step 456 the timing generator 404 is reset to synchronize to the first active carrier and scattered pilot phase. The signal pilot_lock 412 is then set high at step 458, indicating that the pilots have been located successfully.

In a tracking mode of operation, shown as step 460, the scattered pilot search is repeated periodically, and evaluated at decision step 462. This can be done at each cycle of operation, or less frequently, depending upon propagation conditions. The predicted movement of the scattered pilot correlation peak is reflected by appropriate timing in the timing generator 404, and can be used as a test that timing has remained synchronized. Failure of the test at decision step 462 is reported at step 464, and the signal pilot_lock 412 is set low.

It will be appreciated that after the scattered pilots have been located, the task of locating the continual pilots is simplified considerably. As the continual pilots are inserted at a known sequence of positions, the first of which is offset by a multiple of 3 positions with respect to start of the frame, as specified by the ETS 300 744 telecommunications standard. Two of three possible location sets in the data space can therefore be immediately excluded, and it is only necessary to search the third set. Accordingly the continual pilot search is repeated, each iteration beginning at a location 3 carriers higher. New accumulated values and the current start location are stored if they are larger than the previous accumulated value. This is repeated until all continual pilot start positions have been searched. The carrier corresponding to the largest peak value stored will be the first active carrier in the symbol. It is unnecessary to evaluate the "quality" of the continual pilot correlation peak. The scattered pilot search represents a correlation of 142 samples, and has higher noise immunity that of the search for 45 continual pilots. The continual pilot search is almost certain to be succeed if scattered pilot search completed successfully.

The above sequences locate scattered pilot positions within 1/4 symbol period, assuming accumulation at 40MHz, and locate continual pilots in less than 1 symbol period (45 x 115 clock cycles assuming 40MHz operation).

The I and Q data is provided to the pilot locate block 408 by the FFT calculation circuitry 168 (Fig. 14) in bit-reversed order on line 416. This complicates the problem of utilizing a minimum amount of RAM while computing the correlations during pilot localization. Incoming addresses are therefore bit reversed, and computed modulo 12 in order to determine which of 12 possible bins is to store the data. In order to avoid the square root function needed to approximate the carrier amplitude, the absolute values of the data are summed instead as a practical approximation. The scattered pilots are

determined "on the fly". The continual pilots are located on frames which succeed the frames in which the scattered pilots were located.

The operation of the timing generator 404 is now disclosed in further detail. The addressing sequence for the RAM buffer 406 is synchronized by a symbol pulse from the FFT calculation circuitry 168 (Fig. 14). The FFT calculation process runs continuously once the first symbol from has been received following FFT Window acquisition. Addressing alternates between bit-reversed and linear addressing for successive symbols. The timing generator 404 also generates all read-write timing pulses.

Signals u_symbol 466 and c_symbol 468 are symbol timing pulses indicating the start of a new uncorrected symbol or corrected symbol. The signal u_symbol 466 is delayed by latency of the interpolating filter 470 and the compiex multiplier 472, which are synchronized to RAM Address Sequence Timing.

For carrier timing the signals c_carrier0 474, pilot timing signals us_pilot(+) 476, uc_pilot(+) 478, c_tps_pilot(*) 480 and odd_symbol pulse 482 are referenced to a common start pulse sequence. A base timing counter (not shown) is synchronized by the pilot locate sync timing pulse 484, and is therefore offset from symbol timing. Pilot timing outputs are also synchronized to uncorrected symbol output from the buffer 406 or the corrected symbol output delayed by the interpolating filter 470 and the complex multiplier 472. On assertion of the signal resync 204 all timing output is set to inactive states until the first symbol is received. Let the transmitted pilot at carrier k be P_k and the received pilot be P'_k.

$$P'_k = H_k \cdot w_k \cdot P_k \qquad (40)$$

where P_k is described below, and

$$P'_k = I_k + jQ_k \qquad (41)$$

where k indexes pilot carriers, H_k is the channel response and w_k is the reference sequence. We interpolate H_k to generate compensation values for the received data carriers, D'_k:

$$D'_k = I_k + jQ_k \qquad (42)$$

$$D_k = \frac{D'_k}{H_k} \qquad (43)$$

where k indexes data carriers. Received pilots can be demodulated using a locally generated reference sequence and are then passed to the interpolating filter.

The interpolating filter 470, realized in this embodiment with 6 taps and 12 coefficients, is utilized to estimate the portion of the channel between the scattered pilots. As explained above pilots are transmitted at known power levels relative to the data carriers and are modulated by a known reference sequence according to the ETS 300 744 telecommunications standard. The transmitted pilot carrier amplitudes are ± 4/3 of nominal data carrier power (+4/3 for reference bit of 1, -4/3 for the reference bit of 0; quadrature component = 0 in both cases). Interpolation coefficients are selected from the 0-11 cyclic count in the timing generator 404 synchronized to data availability. Appropriate correction factors may be selected for data points to provide on-the-fly correction. The coefficients vary depending on scattered pilot phase. Since the positions of reference pilots vary, therefore coefficients to compensate a given data carrier also vary

The input and output signals, and signals relating to the microprocessor interface 142 of the channel estimation and correction block 170 are described in tables 18, 19 and 20 respectively. The circuitry of the channel estimation and correction block 170 is disclosed in Verilog code listings 18 and 19.

TPS Sequence Extract

The tps sequence extract block 172 (Fig. 14), although set out as a separate block for clarity of presentation, is in actuality partially included in the channel estimation and correction block 170. It recovers the 68-bit TPS data carried in a 68-symbol OFDM frame, and is shown in further detail in Fig. 38. Each bit is repeated on 17 differential binary phase shift keyed ("DBPSK") modulated carriers, the tps pilots, within a COFDM symbol to provide a highly robust transport channel. The 68-bit tps sequence includes 14 parity bits generated by a BCH code, which is specified in the ETS 300 744 telecommunications standard. Of course appropriate modifications can be made by those skilled in the art for other standards having different BCH encoding, and for modes other than 2K mode.

A clipper 486 clips incoming corrected spectrum data to ±1. The sign bit can be optionally evaluated to obtain the clipped result. In comparison block 488 clipped received tps pilot symbols are compared against a reference sequence input. In the described embodiment a value of 0 in the reference sequence matches -1 in the pilot, and a value of 1 in the reference sequence matches +1 in the pilot. Majority vote comparisons are used to provide an overall +1 or -1 result . A result of +1 implies the same modulation as the reference sequence, and a result of -1 implies inverse modulation.

The DBPSK demodulator 490 converts the +/-1 sequence from the majority vote form to a binary form. The sequence converts to a value of 0 if the modulation in current

and previous symbols was the same, and to 1 if modulation between successive symbols is inverted.

From an uninitialized condition a search for either of two sync words in 68-bit tps sequence (4 x 68-bit = 1 superframe) is conducted in the frame synchronizer block 492. The synchronization words of a superframe are as follows:

0011010111101110 sync word for frames 1 and 3

1100101000010001 sync word for frames 2 and 4

Having acquired either sync word, a search for the other is conducted in the appropriate position in the next OFDM frame. On finding the second sync word synchronization is declared by raising the signal tps_sync 494. Data is then passed to the BCH decoder 496, which operates on 14 parity bits at the end of an OFDM frame against received data in the frame. Errors are corrected as necessary.

Decoded data is provided to output store block 498, which stores tps data that is found in a full OFDM frame. The output store block 498 is updated only at the end of an OFDM frame. Only 30 bits of interest are made available. Presently some of these bits are reserved for future use. The length indicator is not retained.

The BCH decoder 496 has been implemented in a manner that avoids the necessity of performing the Berlekamp Algorithm and Chien Search which are conventional in BCH decoding. The Galois Field Multiplier used in the BCH decoder 496 is an improvement of the Galois Field Multiplier which is disclosed in our copending U.S. Application No. 08/801,544.

The particular BCH code protecting the tps sequence is specified in the ETS 300 744 telecommunications standard as BCH (67,53,t=2), having a code generator polynomial

$$h(x) = x^{14} + x^9 + x^8 + x^6 + x^5 + x^4 + x^2 + x + 1 \qquad (44)$$

or equivalently

$$h(x) = (x^7 + x^3 + 1)\ (x^7 + x^3 + x^2 + x + 1) \qquad (45)$$

The left factor is used to generate the Galois Field which is needed for error detection. Referring to Fig. 39, this is calculated in syndrome calculation block 500 which can be implemented using a conventional feedback shift register to generate the α values. The first three syndromes are then computed by dividing the received signal R(x) by the values α^1, α^2, and α^3, again using a conventional feedback shift register implementation, as is well known in the art of BCH decoding. It can be shown that the syndromes are

$$S_0 = (\alpha^1)^{e_0} + (\alpha^1)^{e_1} \quad (46)$$

$$S_1 = (\alpha^2)^{e_0} + (\alpha^2)^{e_1} \quad (47)$$

$$S_2 = (\alpha^3)^{e_0} + (\alpha^3)^{e_1} \quad (48)$$

During the syndrome computation the syndromes are stored in storage registers R[2:0] 502.

In the event S_0 is 0, then it can be immediately concluded that there are no errors in the current tps sequence, and a signal is asserted on line 504 which is provided to error detect block 506, and the data of the received signal R(x) either output unchanged or toggled according to the output of the error detect block 506 on line 508. As explained below, if

$$S_1 \odot S_0 = S_2 \quad (49)$$

then exactly one error is present, a condition which is communicated to the error detect block 506 on line 510. Otherwise it is assumed that two errors are present. More than two errors cannot be detected in the present implementation.

In order to solve the system of three non-linear equations shown above, data flow from the registers R[2:0] 502 into search block 512 is enabled by a signal EOF 514, indicating the end of a frame. Three feedback shift registers 516, 518, 520 having respective Galois Field multipliers 522, 524, 526 for α^{-1} - α^{-3} in the feedback loop are initialized to 50H, 20H, and 3dH (wherein the notation "H" refers to hexadecimal numbers). The feedback shift registers 516, 518, 520 are clocked each time a new data bit is available. The syndromes and outputs of the feedback shift registers 516, 518, 520 are clocked into to a search module, which performs a search for the error positions using an iterative substitution search technique, which will now be described. The outputs of feedback shift registers 516, 518 are multiplied in a Galois Field Multiplier 528.

Considering the case of one error, S_0 is added, modulo 2, preferably using a network of XOR gates 530, to the output of the first feedback shift register 516 (α-gen_0). If the relationship

$$(S_0 + \alpha_{gen_0}) = 0 \quad (50)$$

holds, it is concluded that there is an error in the present data bit. The bit being currently output from the frame store is toggled. The search is halted, and the data is output from the frame store.

Considering the case of two errors, if the following relationship holds, there is an error in the current bit being output from the frame store:

$$(S_0 + \alpha_{gen_0}) \odot (S_1 + \alpha_{gen_1}) = (S_2 + \alpha_{gen_2}) \quad \textbf{(51)}$$

It is now necessary to store the three terms calculated in the immediately preceding equation into the registers R[2:0] 502 which previously stored the syndromes S_0 - S_2. This is represented by line 532.

The process continues, now looking for the second error, and reusing the data in registers R[2:0] 502, which now contains the syndromes as adjusted by the previous iteration. The adjusted syndromes are denoted S_0' - S_2'.

$$S_0' = (S_0 + \alpha_{gen_0}) \quad \text{,etc.} \quad \textbf{(52)}$$

Now, if

$$(S_0' + \alpha_{gen_0}) = 0 \quad \textbf{(53)}$$

the second error has been found, and the bit being currently output from the frame store is toggled by XOR gate 534. If the search fails, more than two errors may be present and an error signal (not shown) is set.

the Galois Field Multiplier 528 is a clocked digital circuit and is disclosed with reference to Fig. 40. The tps data is received very slowly, relative to the other processes occurring in the multicarrier digital receiver 126. It is thus possible to execute the iterative substitution search slowly, and the Galois Field Multipliers are designed for minimum space utilization. They do not require alpha generators, but rely on small constant coefficient multipliers, with iterative feedback to produce the required alpha values. The arrangement takes advantage of the relationship in Galois Field arithmetic

$$\alpha^n = \alpha^1 \cdot \alpha^{n-1} \quad \textbf{(54)}$$

After initialization by a signal init 536 which selects multiplexers 538, 540, the multiplicand A 542 is accumulated in register 544 and repeatedly multiplied by the value α^1 in multiplier 546. The output on line 548 is repeatedly ANDed bitwise with the multiplicand B held in a shift register 550. The output of the shift register is provided on a one bit line 552 to the gate 554. The output of the gate 554 is accumulated in register 556 using the adder 558.

The input and output signals and signals relating to the microprocessor interface 142 of the tps sequence extract block 172 are described in tables 21, 22, and 23. Circuitry of the tps sequence extract block 172 and the BCH decoder 496 is disclosed in Verilog code listings 20 and 21.

Automatic Fine Frequency Control and Automatic Sampling Rate Control

A non ideal oscillator present in the transmission chain of an orthogonal frequency division multiplexed ("OFDM") signal affects all carriers in the OFDM symbols. The OFDM carriers adopt the same phase and frequency disturbances resulting from the noisy local oscillator. Variations in the frequency of the Local Oscillator lead to phase shifts, and consequent loss of orthogonality within the OFDM symbol. Therefore competent automatic frequency control is required in the receiver to track the frequency offsets relative to the transmitter in order to minimize these phase shifts and hence maintain orthogonality.

All the carriers within an OFDM symbol are equally affected by the phase shifts. This is similar to the common phase error caused by phase noise. The common phase error present on all carriers is used to generate an Automatic Frequency Control ("AFC") signal, which is completely in the digital domain, since I/Q demodulation is performed in the digital domain. The approach taken is the calculation of the common phase error for every OFDM symbol. This is achieved by using the reference pilots. The change in the common phase error is measured over time to detect a frequency offset and is used to derive the AFC control signal. The generic approach for the AFC control loop and the automatic sampling rate control loop disclosed below is illustrated in Fig. 41.

Automatic sampling rate control is required when the receiver's master clock is not aligned with that of the transmitter. The misalignment causes two problems: (1) the demodulating carriers have incorrect spacing; and (2) the interval of the FFT calculation is also wrong.

The effect of this timing error is to introduce a phase slope onto the demodulated OFDM data. This phase slope is proportional to the timing error. The phase slope can be determined by calculating the phase difference between successive OFDM symbols, using reference pilots, and estimating the slope of these phase differences. A least squares approach is used for line fitting. The ASC signal is low-pass filtered and fed back to the sinc interpolator 158 (Fig. 13).

The mean phase difference between the reference pilots in subsequent OFDM symbols is used to calculate the frequency deviation. Assuming that the frequency deviations of the local oscillator are constant, then the phase rotates with α, where $\alpha = 2\pi f_d m T_t$ rads. Here f_d is frequency deviation, m is the number of symbols between repetitions of identical pilot positions, and T_t is the period comprising the sum of the active interval and the guard interval. The AFC signal is generated over time by low pass filtering α. The value of the frequency deviation is then used to control the IQ demodulator 144 (Fig. 13).

The AFC and ASC control signals are effective only when a guard interval is passing indicated by the assertion of signal IQGI on line 154 (Fig. 13). This prevents a symbol from being processed under two different conditions.

The correction circuitry 174 (Fig. 14) is shown in greater detail in Fig. 42. Frequency error values output on line 560 are calculated by determining the average of the differences of phase values of corresponding pilots in a current symbol and the previous symbol. The resulting frequency error value is filtered in low pass filter 562 before being fed-back to the IQ demodulator 144 (Fig. 13). It is optional to also evaluate continual pilots in order to cope with larger frequency errors. Sampling rate error, output on line 564 is determined by looking at the phase difference between pilots in a symbol and the same pilots in a previous symbol. The differences vary across the symbol, giving a number of points through which a line can be fitted using the well known method of least squares regression. The slope of this line is indicative of the magnitude and direction of the sampling rate error. The sampling rate error derived in this way is filtered in low pass filter 566 before being fed back to the sinc interpolator 158 (Fig. 13).

A separate store 568 for the scattered pilots contained in 4 symbols is shared by the frequency error section 570 and the sampling rate error section 572. Direct comparison of scattered pilot symbols is thereby facilitated, since the scattered pilot phase repeats every four symbols. In an alternate embodiment where scattered pilots are used to provide control information, storage must be provided for four symbols. In the preferred embodiment, wherein control information is derived from continual pilots, storage for only one symbol is needed.

Recovery of the angle of rotation α from the I and Q data is accomplished in the phase extract block 574, where

$$\alpha = \tan^{-1}(Q/I) \qquad \textbf{(55)}$$

In the presently preferred embodiment, the computations are done at a resolution of 14 bits. The phase extract block 574 is illustrated in greater detail in Fig. 43. The quadrant of α is first determined in block 576. The special cases where I or Q have a zero magnitude or I = Q is dealt with by the assertion of signals on lines 578. If the magnitude of Q exceeds that of I, quotient inversion is accomplished in block 580, utilizing a control signal 582. A positive integer division operation is performed in division block 584. Although this operation requires 11 clock cycles, there is more than enough time allocated for phase extraction to afford it. The calculation of the arctangent of the quotient is accomplished by a pipelined, truncated iterative calculation in block 586of the Taylor Series

$$\tan^{-1}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \ldots, \qquad |x| < 1 \qquad (56)$$

Block 586 is shown in greater detail in the schematic of Fig. 44. The value x^2 is calculated once in block 588 and stored for use in subsequent iterations. Powers of x are then iteratively computed using feedback line 590 and a multiplier 592. The divisions are calculated using a constant multiplier 594 in which the coefficients are hardwired. The sum is accumulated using adder/subtractor 596. The entire computation requires 47 - 48 clock cycles at 40 MHz.

Turning again to Fig. 43, quadrant mapping, and the output of special cases is handled in block 598 under control of block 576. It may be noted that the square error of the result of the Taylor Expansion rises rapidly as α approaches 45 degrees, as shown in Fig. 45 and Fig. 46, which are plots of the square error at different values of α of the Taylor expansion to 32 and 31 terms respectively. The Taylor expansions to 31 and 32 terms are averaged, with the result that the square error drops dramatically, as shown in Fig. 47. A memory (not shown) for holding intermediate values for the averaging calculation is provided in block 598.

Constant Phase Error across all scattered Pilots is due to frequency offset at IQ Demodulator. Frequency Error can be defined as:

$$f_{err} = \frac{\alpha}{2\pi m T_t} \qquad (57)$$

where α, m and T_t have the same meanings as given above. α is determined by taking the average of the difference of phase values of corresponding pilots between the current symbol and a symbol delayed for m symbol periods. In the above equation, m = 1 in the case of continual pilots. This computation uses accumulation block 600 which accumulates the sum of the current symbol minus the symbol that preceded it by 4. Accumulation block 602 has an x multiplier, wherein x varies from 1 to a minimum of 142 (in 2K mode according to the ETS 300 744 telecommunications standard). The low pass filters 562, 566 can be implemented as moving average filters having 10 - 20 taps. The data available from the accumulation block 602 is the accumulated total of pilot phases each sampled m symbols apart. The frequency error can be calculated from

$$f_{err} = \frac{Acc\{new - old\}}{(N)(2)\pi m T_t} \qquad (58)$$

N = 142 in the case of scattered pilots, and 45 for continual pilots, assuming 2K mode of operation according to the ETS 300 744 telecommunications standard. The

technique for determining sampling rate error is illustrated in Fig. 48, in which the phase differences of pilot carriers, computed from differences of every fourth symbol (S_n - S_{n-4}) are plotted against frequency of the carriers. The line of best fit 604 is indicated. A slope of 0 would indicate no sampling rate error.

Upon receipt of control signal 606 from the pilot locate block 408 (Fig. 14), a frequency sweep is initiated by block 608, which inserts an offset into the low-pass filtered frequency error output using adder 610. Similarly a frequency sweep is initiated by block 612, which inserts an offset into the low-pass filtered sampling rate error output using adder 614. The frequency sweeps are linear in increments of 1/8 of the carrier spacing steps, from 0 - 3.5kHz corresponding to control signal values of 0x0-0x7.

A preferred embodiment of the correction circuitry 174 (Fig. 14) is shown in greater detail in Fig. 49. Continual pilots rather than scattered pilots are held in a memory store 616 at a resolution of 14 bits. The generation of the multiplier x for the computation in the accumulation block 618 is more complicated, since in accordance with the noted ETS 300 744 telecommunications standard, the continual pilots are not evenly spaced as are the scattered pilots. However, it is now only necessary to evaluate 45 continual pilots (in 2K mode according to the ETS 300 744 telecommunications standard). In this embodiment only the continual pilots of one symbol need be stored in the store 616. Inclusion of the guard interval size, is necessary to calculate the total duration of the symbol T_t, is received from the FFT window circuitry (block 166, Fig. 14) on line 620.

The input and output signals and signals relating to the microprocessor interface 142 of the circuitry illustrated in Fig. 42 are described in tables 24, 25, 26, and Table 27 respectively. The circuitry is further disclosed in Verilog code listings 24 - 35.

Demapper

The demapping circuitry 176 (Fig. 15) is shown as a separate block for clarity, but in practice is integrated into the channel estimation and correction circuitry. It converts I and Q data, each at 12-bit resolution into a demapped 12-bit coded constellation format (3-bit I, I soft-bit, 3-bit Q, Q soft-bit). The coded constellation is illustrated in Fig. 50 and Fig. 51. For 64-QAM the 3 bits are used for the I and Q values, 2 bits for 16-QAM 2-bits and 1 bit for QPSK.

For example in Fig. 51 values of I= 6.2, Q= -3.7 would be demapped to: I-data = 001; I soft-bit=011; Q-data=101; Q soft-bit=101.

The input and output signals of the demapping circuitry 176 are described in tables 28 and 29 respectively.

Symbol Deinterleaver

The symbol deinterleaver 182 (Fig. 15) reverses the process of symbol interleaving of the transmitted signal. As shown in Fig. 52 the deinterleaver requires a 1512 x 13

memory store, indicated as block 622. The address generator 624 generates addresses to write in interleaved data and read out data in linear sequence. In practice the address generator 624 is realized as a read address generator and a separate write address generator. Reading and writing occur at different instantaneous rates in order to reduce the burstiness of the data flow. The address generator 624 is resynchronized for each new COFDM symbol by a symbol timing pulse 626. Carrier of index 0 is marked by carrier0 pulse 628. Addresses should be generated relative to the address in which this carrier is stored.

The input and output signals of the symbol deinterleaver 182 are described in tables 30 and 31 respectively. Circuitry of the symbol deinterleaver 182 is disclosed in Verilog code listing 22.

Bit Deinterleaver

Referring to Fig. 54, the bit deinterleaver 184 (Fig. 15) reverses the process of bit-wise interleaving of the transmitted signal, and is shown further detail in Fig. 53. In soft encoding circuitry 630 input data is reformatted from the coded constellation format to a 24 bit soft I/Q format. The soft encoding circuitry 630 is disclosed for clarity with the bit deinterleaver 184, but is realized as part of the symbol deinterleaver discussed above. The deinterleave address generator 632 generates addresses to read the 6 appropriate soft-bits from the 126 x 24 memory store 634, following the address algorithm in the ETS 300 744 telecommunications standard. The deinterleave address generator 632 is resynchronized for each new COFDM symbol by the symbol timing pulse 626.

The output interface 636 assembles I and Q output data streams from soft-bits read from the memory store 634. Three I soft bits and three Q soft bits are extracted from the memory store 634 at each deinterleave operation, and are parallel-serial converted to provide the input data stream to the Viterbi Decoder 186 (Fig. 15).

The input and output signals of the bit deinterleaver 184 are described in tables 32 and 33 respectively. Circuitry of the bit deinterleaver 184 is disclosed in Verilog code listing 23.

Host Microprocessor Interface

The function of the microprocessor interface 142 is to allow a host microprocessor to access control and status information within the multicarrier digital receiver 126 (Fig. 12). The microprocessor interface 142 is shown in greater detail in Fig. 55. A serial interface 638 and a parallel interface 640 are provided, the latter being primarily of value for testing and debugging. The serial interface 638 is of known type and is I2C compatible. The microprocessor interface 142 includes a maskable interrupt capability allowing the receiver to be configured to request processor intervention depending on

internal conditions. It should be noted, that the multicarrier digital receiver 126 does not depend on intervention of the microprocessor interface 142 for any part of its normal operation.

The use of interrupts from the point of view of the host processor is now described. "Event" is the term used to describe an on-chip condition that a user might want to observe. An event could indicate an error condition or it could be informative to user software. There are two single bit registers (not shown) are associated with each interrupt or event. These are the condition event register and the condition mask register.

The condition event register is a one bit read/write register whose value is set to one by a condition occurring within the circuit. The register is set to one even if the condition only existed transiently. The condition event register is then guaranteed to remain set to one until the user's software resets it, or the entire chip is reset. The condition event register is cleared to zero by writing the value one. Writing zero to the condition event register leaves the register unaltered. The condition event register must be set to zero by user software before another occurrence of the condition can be observed.

The condition mask register is a one bit read/write register which enables the generation of an interrupt request if the corresponding condition event register is set. If the condition event is already set when 1 is written to the condition mask register an interrupt request will be generated immediately. The value 1 enables interrupts. The condition mask register clears to zero on chip reset. Unless stated otherwise a block will stop operation after generating an interrupt request and will restart soon after either the condition event register or the condition mask register are cleared.

Event bits and mask bits are always grouped into corresponding bit positions in consecutive bytes in the register map. This allows interrupt service software to use the value read from the mask registers as a mask for the value in the event registers to identify which event generated the interrupt. There is a single global event bit that summarizes the event activity on the chip. The chip event register presents the OR of all the on-chip events that have 1 in their respective mask bit. A value of 1 in the chip mask bit allows the chip to generate interrupts. A value of 0 in the chip mask bit prevents any on-chip events from generating interrupt requests. Writing 1 or 0 to the chip event register has no effect. The chip event register only clears when all the events enabled by a 1 in their respective mask bits have been cleared.

The IRQ signal 642 is asserted if both the chip event bit and the chip event mask are set. The IRQ signal 642 is an active low, "open collector" output which requires an

off-chip pull-up resistor. When active the iRQ output is pulled down by an impedance of 100Ω or less. A pull-up resistor of approx. 4kΩ is suitable.

The input and output signals of the microprocessor interface 142 are described in tables 34 and 35 respectively.

System Controller

The system controller 198 (Fig. 15), which controls the operation of the multicarrier digital receiver 126 (Fig. 12), in particular channel acquisition and the handling of error conditions, is shown in further detail in Fig. 56.

Referring to the state diagram in Fig. 57, the channel acquisition sequence is driven by four timeouts.

(1) AGC acquisition timeout. 20 ms (80 symbols) are allowed for the AGC to bring up the signal level, shown in step 644. Then the FFT window is enabled to start acquisition search in block 646.

(2) Symbol acquisition timeout: 200 symbol periods, the maximum guard interval plus active symbol length, is allocated to acquire the FFT window in step 648. Another 35 symbol periods are allocated to pilot location in step 650. Approximately 50 ms are required to process 2K OFDM symbols. An option is provided to exit step 650 as soon as the pilots have been located to save acquisition time in non-extreme situations.

(3) Control Loop Settling timeout: A further 10 ms, representing approximately 40 symbols is allocated to allow the control loops to settle in step 652. An option is provided to exit step 652 and return to an initial step resync 654 if pilots have been lost if control loop settling timeout occurs.

(4) Viterbi synchronization timeout: In block 656 approximately 150 symbol periods are allocated for the worst case of tps synchronization, indicated by step 658 and approximately 100 symbol periods for the Viterbi Decoder 186 (Fig. 15) to synchronize to the transmitted puncture rate, shown as step 660. This is approximately 65 ms. In reasonable conditions it is unnecessary to wait this long. As soon as Viterbi synchronization is established, then transition to the system_lock state 662. It is possible to bypass the tps synchronization requirement by setting parameters (see table below) in the receiver parameters register and setting set_rx_parameters to 1.

If acquisition fails at any stage, the process automatically returns to step resync 654 for retry.

Having acquired lock, the system will remain in lock unless a Reed-Solomon overload event occurs, i.e. the number of Reed-Solomon packets with uncorrectable errors exceeds a predetermined value (the rso_limit value) in any 1 second period. If any of the 4 synchronizing state machines in the acquisition sequence, FFT window (step 648), pilot locate (step 650), tps synchronization (step 658) and Viterbi synchroni-

zation (step 660), lose synchronization once channel acquisition has occurred, no action will be taken until an event, rso_event, occurs and the step resync 654 is triggered automatically.

In poor signal conditions acquisition may be difficult, particularly the Viterbi synchronization. Therefore a bit is optionally provided in the microprocessor interface 142 (Fig. 12), which when set extends the timeouts by a factor of 4.

The input and output signals, and the microprocessor interface registers of the system controller 198 are described in tables 36, 37, 38, and 39 respectively.

Tables

Pin Name	I/O	Description
Tuner/ADC Interface		
SCLK	O	Sample clock for ADC
IDATA[9:0]	I	Input ADC data bus (10-bit)
AGC	O	Automatic Gain Control to tuner(Sigma-Delta output)
XTC[2:0]	O	External Tuner Control Outputs
MPEG-2 Transport Interface		
OUTDAT[7:0]	O	MPEG-2 Transport Stream Data bus
OUTCLK	O	MPEG Transport Stream Output Clock
SYNC	O	MPEG Transport Stream Sync pulse (1 per 188byte)
VALID	O	MPEG Transport Stream Valid data flag
ERROR	O	MPEG Transport Stream Errored data flag
Serial Host Microprocessor Interface		
SD	I/O	Serial Interface Data
SC	I	Serial Interface Clock
SDT	I/O	Serial Data Through
SCT	O	Serial Clock Through (40MHz clock out when DEBUG is high)
SADDR[2:0]	I	Serial Address Inputs (Hardwired external value) used as TSEL pins when DEBUG is high
Parallel Host Microprocessor Interface		

Pin Name	I/O	Description
MA[5:0]	I	Microprocessor Address Bus
MD[7:0]	I/O	Microprocessor Data Bus 2-bit/DEBUG data @40MHz
MWE	I	Microprocessor Write Enable
MCE	I	Microprocessor Chip Enable
NOTIRQ	O	Interrupt Request
JTAG Test Access Port		
TCK	I	JTAG Test Clock
TMS	I	JTAG Test Mode Select
TDI	I	JTAG Test Data In
TDO	O	JTAG Test Data Out
NTRST	I	JTAG TAP Controller Reset
Miscellaneous Pins		
NRESET	I	Asynchronous Reset
CLK40	I	40MHz Input Clock
TSTRI	I	Transport Stream Interface tristate control
TA (MA[6])	I	Test Address Bit - Snooper access (Bit 7 of up address bus)
DEBUG	I	Test Pin
TSEL [2:0]/SADDR[2:0]	I	Internal Test Inputs (mux out internal data onto MD[7:0]) 0 = normal upi, 1= fft input data (24-bit), 2 = fft output data (24-bit), 3 = channel correction output data (24-bit), 4 = fec input data.(2 x 3-bit softbit) all data clocked out @40MHz, 24-bit data in 4 bytes. Clock brought out on SCT pin, for convenience. Symbol timing/other synch. signals indicated with market bits in data.
TLOOP	I	Test Input

Table 4

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x00	**Event Reg.**			
	0	R/W/0	chip_event	OR of all events which are interrupt-enabled (unmasked)
	1	R/W/0	lock_failed_event	Set to 1 if channel acquisition sequence fails
	2	R/W/0	rs_overload_event	Set to 1 if Reed-Solomon Decoder exceeds set threshold within one 1 second period
0x01	**Mask Reg.**			
	0	R/W/0	chip_mask	Set to 1 to enable IRQ output
	1	R/W/0	lock_failed_mask	Set to 1 to enable interrupt on channel acquisition fail
	2	R/W/0	rs_overload_mask	Set to 1 to enable interrupt on RS error threshold exceeded
0x02	**Status Reg.**			
	0	R/0	system_locked	Set to 1 when system acquired channel successfully
	1	R/0	viterbi_sync	Set to 1 when Viterbi is synchronized
	2	R/0	tps_sync	Set to 1 when OFDM frame carrying TPS data has been synchronized to.
	3	R/0	pilot_loc	Set to 1 when pilots in COFDM symbol have been located and synchronized to
	4	R/0	fft_loc	Set to 1 when guard interval has been located and synchronized to.
	7:5	R/1	viterbi_rate	Received Viterbi Code rate
0x04-0x05	**Control Reg:**			
	0	R/W/0	change_channel	When set to 1, holds device in "Reset" state. Clearing this bit initiates channel change.

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	1	R/W/0	agc_invert	Invert AGC Signa-Delta output. Default setting means low output associated with reduced AGC gain.
	2	R/W/0	o_clk_phase	Set to 1 to invert phase of output clock. Default condition: output data changes on falling edge of output clock.
	3	R/W/0	set_rx_parameters	Set to 1 to take Receiver Parameter Data from Receiver Parameter Register. Default condition: settings taken from TPS data (longer channel acquisition time)
	4	R/W/0	extend_agc	Set to 1 to hold acquisition sequence in agc_acquire state
	5	R/W/0	extend_fs	Set to 1 to hold acquisition sequence in fs_acquire state
	6	R/W/0	extend_settle	Set to 1 to hold acquisition sequence in fs_settle state
	7	R/W/0	extend_sync	When set to 1 to hold acquisition sequence in vit_sync state
	10:8	R/W/0	xtc	External Tuner Control bits (external pins XTC[2:0])
	11	R/W/0	i2c_gate	12C "Gate" signal; setting this to 1 enables the isolation buffer between the "processor side12C" bus and the "Tuner side" 12C so the processor can access a Tuner through COFDM device. Setting to 0 closes the "gate" to prevent 12C bus noise affecting delicate RF.

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	12	R/W/ (TSTRI)	ts_tri	Transport Stream Tristate control - set to 1 to tristate MPEG TS interface (e.g. to mux a QPSK device to same MPEG demux). Power-on state of TS output controlled by external pin TSTRI.
	13	R/W/0)	fast_ber	Set to 1 to reduce BER counter, vit_ill_state counter and rso_counter, counter periods from 1 sec to 100ms.
	15	R/W/0	soft_reset	Software Reset - set to 1 to reset all blocks except upi. Set to 0 to release.
0x06-0x07	Receiver Parameter Register:			
	15:14	R/W/2	upi_constellation	Constellation Pattern for Demapper and Bit Deinterleaver (reset condition = 64-QAM)
	13:12	R/W/0	upi_guard	Guard Interval: 00 = 1/32, 01 = 1/16, 10 = 1/8, 11 = 1/4
	11:9	R/W/0	upi_alpha	Hierarchical Tranmission Mode or "alpha value" (reset condition = non-hierarchical mode)
	7:5	R/W/0	upi_hp_rate	Viterbi Code Rate for HP stream - in non-hierarchical mode this is taken as the Viterbi Code Rate (reset condition = 1/2 rate code)
	4:2	R/W/0	upi_lp_rate	Viterbi Code Rate for LP stream (reset condition = 1/2 rate code)
	1:0	R/W/0	upi_tx_mode	Tranmission mode (00=2K, 01=8K, others reserved)
0x08	7:0	R/W/0	rso_limit	Errored packet per second limit (for rs_overload_event bit)

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x09	7:0	R/0	rso_count	Count of Uncorrectable Transport Packets per second (saturates at 255). Write to register to latch a stable count value which can then be read back
0x0a-0x0b	15:0	R/0	ber	BER (before RS) deduced from RS corrections in 1 second period - max correctable bit errors ~1.35M/sec for 7/8, 64-QAM, 1/32 GI (equivalent to 43.e-3 BER assuming useful bitrate of 31.67 e6). Only top 16 bits of 21 bit counter are visible - resolution of ~1e-6 depending on code-rate, constellation GI length. Write to register to latch a stable count value which can then be read back.
0x0c-0x0d	15:0	R/0	agc_level	AGC "Control Voltage" (msb's)
0x0e-0x0f	11:0	R/0	freq_error	IQ Demodulator Frequency Error (from feedback loop)
0x10-0x13	TPS Data (including future use bits)			
	1:0	R/0	tps_frame	Number of last received cmplete OFDM frame in superframe
	3:2	R/0	tps_constellation	Constellation Pattern from TPS data
	7:5	R/0	tps_alpha	Hierachical Transmission Information
	10:8	R/0	tps_hp_rate	Viterbi Code Rate of High-Priority stream (In non-hierarchical mode this is the code rate of the entire stream)
	13:11	R/0	tps_lp_rate	Viterbi Code Rate of Low-Priority stream
	15:14	R/0	tps_guard_int	Guard Interval

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	17:16	R/0	tps_tx_mode	Transmission Mode
	31:19	R/0	tps_future	Undefined bits allocated for future use
*** Debug Access ***				
0x20-0x21	15	R/W/0	agc_open	Set to 1 to break AGC control loop
	11:0	R/W/0	agc_twiddle	AGC twiddle factor
0x22-0x23		R/W/0	agc_loop_bw	AGC Control loops parameters
0x24-0x25	15	R/W/0	freq_open	Set to 1 to break freq control loop
	14	R/W/0	freq_nogi	Set to 1 to allow frequency update anytime, not just during Guard Interval
	11:0	R/W/0	freq_twiddle	IQ Demod twiddle factor
0x26-0x27			freq_loop_bw	Frequency Control Loop parameters
0x28-0x29	15	R/W/0	sample_open	Set to 1 to break sample control loop
	14	R/W/0	sample_nogi	Set to 1 to allow sample update anytime, not just during Guard Interval
	11:0	R/W/0	sample_twiddle	Sampling Rate Twiddle factor
0x2a-0x2b		R/W/0	sample_loop_bw	Sampling Rate Control Loop parameters
0x2c-0x2d	11:0	R/0	sampling_rate_err	Sampling Rate Error (from feedback loop)
0x30-0x31	15	R/W/0	lock_fft_window	Set to 1 to prevent fft_window moving in Tracking mode
	14	R/W/0	inc_fft_window	Write 1 to move fft_window position one sample period later (one-shot operation)
	13	R/W/0	dec_fft_window	Write 1 to move fft_window position one sample period earlier (one-shot operation)
	12:0	R/0	fft_window	FFT Window position

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	7:0	R/W/0	fft_win_thresh	FFT Window Threshold
0x34-0x35	15	R/W/0	set_carrier_0	Set to 1 to use carrier_0 value as setting
	11:0	R/W/0	carrier_0	Carrier 0 position; readback value detected by Pilot Locate algorithm or force a value by writing over it
0x36	7:0	R/W/	csi_thresh	Channel State Information threshold - the fraction of mean level below which data carriers are marked by a bad_carrier flag. Nominally 0.2 (for 2/3 code rate).
0x37				
0x38-0x39	11:0	R/0	vit_ill_states	Viterbi Illegal State Rate (per second)Write to register to latch count which can then be read back
***** SNOOPERS ***** (External test address bit TA[6] = 1)				
0x40-0x41	15:14 11:0	R/W/R/W	T,IQGIFreq_error[11:0]	IQ Demod Snooper (Note: bit 0 = lsb of highest addressed byte, 21)
0x44-0x47	31:30 27:16 11:0	R/W R/W R/W	T, Valid Q-data[11:0] I-data[11:0]	Low-Pass Filter Snooper
0x48-0x4d	47:46 43:32 31 27:16 11:0	R/W R/W R/W R/W R/W	T,SincGI Sample_err[11:0] Valid Q-data[11:0] I-data[11:0]	Resampler Snooper
0x50-0x53	31:29 27:16 11:0	R/W R/W R/W	T, Valid,Resync Q-data[11:0] I-data[11:0]	FFT Snooper
0x54-0x57	31:30 29:28 27:16 11:0	R/W R/W R/W R/W	T, Valid, Symbol,Resync Q-data[11:0] I-data[11:0]	Channel Estimation & Correction Snooper
0x58-0x5b	31:30 29:28 27:16 11:0	R/W R/W R/W R/W	T, Resync u_symbol, uc_pilot Q-data[11:0] I-data[11:0]	Frequency & Sampling Error Snooper

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x5c-0x5f	31:30 29:28 27:16 15 11:0	R/W R/W R/W R/W R/W	T, Resync c_symbol, tps_pil. Q-data[11:0] reference_seq I-data[11:0]	TPS Sequence Extract Snoopers
0x60-0x65	39 36:35 34:32 27:16 15:14 13 11:0	R/W R/W R/W R/W R/W R/W R/W	T constellation alpha Q-data[11:0] Valid, c_symbol c_carrier0 I-data[11:0]	Demap Snooper
0x68-0x6a	23:22 21:20 19 11:0	R/W R/W R/W R/W	T, valid symbol, carrier0 odd_symbol demap_data[11:0]	Symbol Deinterleave Snooper
0x6c-0x6e	23:21 20:19 18:16 11:0	R/W R/W R/W R/W	T, valid, symbol constellation alpha symdi_data[11:0]	Bit Deinterleaver Snooper
0x70-0x71	15:13 6:4 2:0	R/W R/W R/W	T, valid, resync Q-data[2:0] I-data[2:0]	Viterbi Snooper
0x72-0x73	15:14 13:12 7:0	R/W R/W R/W	T, valid, resync, eop vit_data[7:0]	Forney Deinterleaver Snooper
0x74-0x75	15:14 13:12 7:0	R/W R/W R/W	T, valid, resync, eop deint_data[7:0]	Reed Solomon Snooper
0x76-0x77	15:14 13:12 11:0 7:0	R/W R/W R/W R/W	T, valid, resync, eop error_val, error deint_data[7:0]	Output Interface Snooper
0x78-0x7b	31 30:20 19:18 17 16 14 13:8 6:5 4:3 2:0	R/W R/W R/W R/W R/W R/W R/W R/W R/W R/W	T tps_data[10:0] pkt_err, err_val vit_ill_state vit_ill_val rs_corr_val rs_correct[5:0] vit_sync, tps_sync pilot_loc, fft_loc vit_rate[2:0]	System Controller Snooper

Table 5

Signal	Description
clk	40MHz main clock
clk20M	20MHz sample clock (used as a "valid" signal to indicate when valid input samples are received)
data[9:0]	sampled data input from ADC
agc_resync	control input; held low on channel change - on transition to high AGC should reset itself and accumulate new control voltage for new channel.
lupdata[7:0] (bi-di)	Internal Microprocessor Data bus
upaddr[2:0]	Internal Microprocessor Address Bus (only 2-bits required)
upwstr	Internal uP write strobe
uprstr	Internal uP read strobe
upsel1	Internal Address decode output (high = valid for 0x0c-0x0d)
upsel2	Internal Address decode output (high = valid for 0x20-0x23)
te, tdin	Scan inputs

Table 6

Signal	Description
agc	Signal - Delta modulated output signal; when integrated by external RC it provides an analogue representation of the internal digital "control voltage" valueInterpolated output data
tdout	scan outputs

Table 7

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x0c-0x0d	15:0	R/0	agc_level	AGC "Control Voltage" (msb's)
0x20-0x21	15	R/W/0	agc_open	Set to 1 to break AGC control loop
	11:0	R/W/0	agc_twiddle	AGC twiddle factor
0x22-0x23		R/W/0	agc_loop_bw	AGC Control loops parameters

Table 8

Signal	Description
clk	40MHz main clock
nrst	Active-low synchronous reset
clk20M	20MHz sample clock (used as a "valid" signal to indicate when input data sample is valid)
sample[9:0]	input data sample from ADC. (AGC should ensure that this white-noise-like signal is scaled to full dynamic range)
freq_err[11:0]	Frequency Error input - 1Hz accurate tuning over +/-0.5 carrier spacing
IQGI	Valid pulse for enable frequency error signal. The effect of the frequency control loop is held off until a guard interval is passing through the IQ Demod block. (IQGI is generated by the FFT window and indicates when a guard interval is passing).
te, tdin	Scan test inputs

Table 9

Signal	Description
I-data[11:0]	I data-stream to be low-pass filtered (40 MHZ timing)
Q-data[11:0]	Q data-stream to be low-pass filtered (40 MHZ timing)
valid	Valid output data indicator; high if data is being output on this clock cycle (40 MHZ timing)
tdout	Scan test output

Table 10

Signal	Description
clk	40MHz clock (2x sample clock)
nrst	Active-low synchronous reset
valid_in	high-pulse indicating valid data from IQ-demodulator (40MHz timing)
i_data[11:0], q_data[11:0]	input data from IQ-demodulator (20Msps)
te, tdin	Scan test inputs

Table 11

Signal	Description
i_out[11:0], q_out[11:0]	Low-Pass filtered output data

Signal	Description
valid	Output pulse indicating valid data output (decimated to 10Msps)
tdout	Scan test output

Table 12

Signal	Description
clk40M	40MHz main clock (2x sample clock)
valid_in	input data valid signal; when valid is low, input data should be ignored
i_data[11:0], q_data[11:0]	input data from low-pass filter (decimated to 10Msps)
sr_err[11:0]	SamplingRate Error feedback fro Freq/Sampling Error block
SincGI	Valid pulse for Error signal; effect of Sampling Rate contol loop is held off until guard interval is passing through Sinc Interpolator. FFT Window block generates this signal at appropriate time.
te,tdin	Scan test signals

Table 13

Signal	Description
i_out[11:0], q_out[11:0]	Interpolated output data
valid	Output pulse indicating valid data output)
tdout	Scan test output

Table 14

Signal	Description
clk40M	40MHz clock (2x sample clock)
valid_in	input data valid signal; when valid is low, input data should be ignored
i_data[11:0]	input data from front-end (ignore quadrature data for this block)
resync	Control signal: forces Sync FSM back to acquisition mode when pulsed high
guard[1:0]	Expected guard interval; programmed by Host uP to aid fft window acquisition. 00 = 1/32, 01 = 1/16, 10 = 1/8, 11 = 1/4

Signal	Description
lupdata[7:0] (bi-di)	Internal Microprocessor Data bus (bi-directional)
upaddr[0]	Internal uP address bus (only 1-bit required)
upwstr	Internal uP write strobe
uprstr	Internal uP read strobe
upsel	Address decode output to select FFT window block

Table 15

Signal	Description
FFT_Window	Timing output pulse; low for 2048 samples indicating the active interval
fft_lock	Output pulse indicating status of Sync FSM; 1 = Symbol acquired
rx_guard[1:0]	Received Guard Interval Size: 00 = 1/32, 01 = 1/16, 10 = 1/8, 11 = ¼
IQGI	Timing pulse indicating when the guard interval should arrive at the IQ demodulator (Frequency Error only corrected in the Guard Interval)
SincGI	Timing pulse indicating when the guard interval should arrive at the Sinc Interpolator (Sampling Error only corrected in the Guard Interval)
sr_sweep[3:0]	Sampling Rate sweep output; 4-Bit output used by Frequency and Sampling Error block to generate Sampling Rate "ping-pong" sweep during FFT window acquisition.

Table 16

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x30-0x32	15	R/W/0	lock_fft_window	Set to 1 to prevent fft_window moving in Tracking mode
	14	R/W/0	inc_fft_window	Write 1 to move fft_window position one sample period later (one-shot operation)
	13	R/W/0	dec_fft_window	Write 1 to move fft_window position one sample period earlier (one-shot operation)
	12:0	R/0	fft_window	FFT Window position

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	7:0	R/W/0		

Table 17

Signal	Description
clk40M	40MHz clock (2x sample clock)
nrst	Synchronous reset (active low)
valid_in	input data valid signal; when valid is low, input data should be ignored
i_data[11:0], q_data[11:0]	input data from FFT
symbol	Symbol timing pulse from FFT; high for first valid data value of a new symbol
resync	Resynchronization input triggered on e.g. channel change. Pulsed high to indicate return to acquisition mode (wait for first symbol pulse after resync before beginning pilot search)
lupdata[7:0] (bi-di)	Internal Microprocessor Databus
upaddr[0]	Internal uP address bus (only 1-bit required)
upwstr	Internal uP write strobe
uprstr	Internal uP read strobe
upsel	Internal address decode output: high for addresses 0x032-0x033

Table 18

Signal	Description
ui_data[11:0], uq_data[11:0]	Uncorrected spectrum data, as read from RAM buffer (for Frequency/Sampling Error block)
u_symbol	Uncorrected symbol start; high for first carrier of the uncorrected symbol
us_pilot	high for any carrier which is a scattered pilot in the uncorrected symbol
ci_data[11:0], cq_data[11:0]	Corrected spectrum data; as output from the complex multiplier
valid	high for valid corrected symbol - data carriers only
bad_carrier	high if interpolated channel response for the carrier is below pre-set fraction of the mean of carriers of previous symbol - viterbi will discard the data carried by this carrier

Signal	Description
c_symbol	high for the first carrier in the corrected symbol
c_carrier0	high for the first active carrier in the corrected symbol (a continual pilot corresponding to a carrier index value of 0)
c_tps_pilot	high for any carrier in the corrected symbol which is a TPS pilot
pilot_lock	output high if pilots successfully located at the end of pilot acquisition phase.
odd_symbol	high for symbol period if symbol is odd number in frame (as determined from scattered pilot phase)
c_reference_seq	Reference sequence output to TPS Sequence block
freq_sweep[2:0]	Frequency Sweep control; incrementing 3-bit count which increments IQ Demodulator LO offset in Frequency and Sampling block. Sweeps 0-0.875 carrier spacing offset in 0.125 carrier spacing steps

Table 19

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x32-0x33	15	R/W/0	set_carrier_0	Set to 1 to use carrier_0 value as setting
	11:0	R/W/0	carrier_0	Carrier 0 position
0x36	7:0	R/W/	csi_thresh	Channel State Information threshold - the fraction of mean level below which data carriers are marked by a bad_carrier flag. Nominally 0.2 (for 2/3 code rate). A value of 0 would turn CSI off for comparison testing.
0x37	7:0			

Table 20

Signal	Description
clk40M	40MHz clock (2x sample clock)
ci_data[11:0]	corrected pilot data from Channel Estimation and Correction (only need I data because corrected pilots should only insignificant Im component; - only need sign bit)
tps_pilot	high for single clock cycle when data input is a tps_pilot - use like a valid signal.
reference_seq	Reference Sequence PRBS input from Channel Estimation & Correction - ignore for non-tps_pilot values
c_symbol	timing pulse high for 1 clock cycle for first carrier in new symbol (whether or not that carrier is active)
lupdata[7:0] (bi-di)	Internal Microprocessor Databus
upaddr[1:0]	Internal uP address bus (only 2-bits required)
upwstr	Internal uP write strobe
uprstr	Internal uP read strobe
upsel	Internal address decode output; high for addresses 0x10-0x13

Table 21

Signal	Description
tps_data [29:0]	Output tps data (held static for 1 OFDM frame): tps_data[1:0] = frame number tps_data[3:2] = constellation tps_data[6:4] = hierarchy tps_data[9:7] = code rate, HP stream tps_data[12:10] = code rate, LP stream tps_data[14:13] = guard interval tps_data[16:15] = transmission mode tps_data[29:17] = future use bits Note that parameters are transmitted for the next frame; outputs should be double-buffered so parameters appear at block outputs in the correct frame (used by Demapper and Symbol/Bit deinterleave blocks to decode incoming data)
tps_sync	Status output from Frame Sync FSM - set to 1 when FSM is sync'd i.e when 2 valid sync words have been received in expected postions AND correct TPS data is available at the block outputs.

Table 22

0x10-0x13	TPS Data (including future use bits)			
	1:0	R/0	tps_frame	Number of last received complete OFDM frame in superframe

0x10-0x 13	TPS Data (including future use bits)			
	3:2	R/0	tps_constellation	Constellation Pattern from TPS data
	7:5	R/0	tps_alpha	Hierarchical Transmission Information
	10:8	R/0	tps_hp_rate	Viterbi Code Rate of High-Priority stream (In non-hierarchical mode this is the code rate of the entire stream)
	13:11	R/0	tps_lp_rate	Viterbi Code Rate of Low-Priority stream
	15:14	R/0	tps_guard_int	Guard Interval
	17:16	R/0	tps_tx_mode	Transmission Mode
	31:19	R/0	tps_future	Undefined bits allocated for future use

Table 23

Signal	Description
clk40M	40MHz clock (2x sample clock)
nrst	Active low reset
us_pilot	input data valid signal; high when a scattered pilot is output from the Channel Estimation & Correction block
guard[1:0]	Guard Interval from which symbol period Tt can be deduced:00 = 1/32 (Tt = 231us) , 01 = 1/16 (238us), 10 = 1/8 (252us) , 11 = 1/4 (280us)
ui_data[11:0], uq_data[11:0]	input data from Channel Estimation & Correction (Uncorrected spectrum)
u_symbol	Symbol timing pulse from Channel Estimation & Correction; high for first valid data value of a new symbol (uncorrected spectrum)
resync	Resynchronization input triggered on e.g. channel change. Pulsed high to indicate return to acquisition mode (wait for first symbol pulse after resync before beginning Pilot search)
sr_sweep[3:0]	Sampling Rate Sweep control from FFT Window block: 0 = 0Hz offset, 1=+500Hz, 2=-500Hz,3=+1000Hz, 4=-1000Hz,5=+1500Hz,6=-1500Hz,7=+2000Hz,8=-2000Hz
freq_sweep[3:0]	Frequency Sweep control from Channel Estimation & Correction block; represents number n range 0-7 frequency offset = nx500Hz

Signal	Description
lupdata[7:0] (bi-di)	Internal Microprocessor Databus
upaddr[3:0]	Internal uP address bus (only 4-bit required)
upwstr	Internal uP write strobe
uprstr	Internal uP read strobe
upsel1	Internal address decode output; high for addresses 0x0e-0x0f
upsel2	Address decode for addresses in range 0x24-0x2d

Table 24

Signal	Description
frequency_error	frequecy error output (to IQ Demod)
sampling_rate_error	Sampling Rate Error output (to Sinc Interpolator)
freq_lock	status output; high if frequency error low
sample_lock	status output; high if sampling rate error low

Table 25

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x0e-0x0f	11:0	R/0	freq_error	IQ Demodulator Frequency Error (from feedback loop)

Table 26

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x24-0x25	15	R/W/0	freq_open	Set to 1 to break freq control loop
	14	R/W/0	freq_nogi	Set to 1 to allow frequency update anytime, not just during Guard Interval
	11:0	R/W/0	freq_twiddle	IQ Demod twiddle factor
0x26-0x27			freq_loop_bw	Frequency Control Loop parameters
0x28-0x29	15	R/W/0	sample_open	Set to 1 to break sample control loop

Address (Hex)	Bit No.	Dir/Re-set	Register Name	Description
	14	R/W/0	sample_nogi	Set to 1 to allow sample update anythime, not just during Guard Interval
	11:0	R/W/0	sample_twiddle	Sampling Rate Twiddle factor
0x2a-0x2b		R/W/0	sample_loop_bw	Sampling Rate Control Loop parameters
0x2c-0x2d	11:0	R/0	sampling_rate_err	Sampling Rate Error (from feedback loop)

Table 27

Signal	Description
clk40M	40MHz clock (2x sample clock)
valid_in	input data valid signal; when valid is low, input data should be ignored
i_data[11:0], q_data[11:0]	input data from Channel Estimation & Correction.
bad_carrier_in	Carrier Status falg - set if carrier falls below acceptable level; indicates to viterbi that data from this carrier should be discarded from error correction calculations.
c_symbol	Timing synchronization signal - high for the first data sample in the corrected COFDM symbol.
constellation[1:0]	control signal which defines constellation: 00 = QPSK, 01 = 16-QAM, 10 = 64-QAM
alpha[2:0]	control signal defining hierarchical transmission parameter, alpha: 000 = non-hierarchical transmission, 001 = alpha value of 1, 010 = alpha value of 2, 011 = alpha value of 4 (Note the first release of the chip will not support hierarchical transmission)

Table 28

Signal	Description
out_data[11:0]	deinterleaved output data 6 I, 6 Q format
bad_carrier	bad_carrier flag carried through demap process unchanged.
valid	Valid output data indicator; high if data is being output on this clock cycle

Signal	Description
d_symbol	Symbol timing pulse re-timed to synchronize with out_data

Table 29

Signal	Description
clk40M	40MHz clock (2x sample clock)
valid_in	input data valid signal; when valid is low, input data should be ignored
demap_data[11:0]	input data from Demapper. Data is in 6-bit I, 6-bit Q format (for 64_QAM)
bad_carrier_in	Carrier status signal - set if carrier falls below limits; indicates to viterbi that data should be ignored. Carried with data as extra bit through deinterleaver store.
symbol	Timing synchronization signal - high for the first data sample in a COFDM symbol. Used to resynchronize address generation
carrier0	Timing pulse - high for the first active carrier (corresponding to carrier index value of 0) in a symbol
odd_symbol	high if symbol is odd number in the frame (different interleaving pattern in odd and even symbols within 68-symbol frame)

Table 30

Signal	Description
out_data[11:0]	deinterleaved output data coded constellation format
bad_carrier	Bad carrier output having passed through deinterleave RAM.
valid	Valid output data indicator; high if data is being output on this clock cycle
d_symbol	Output timing synchronization signal - high for first data sample in de-interleaved COFDM symbol.

Table 31

Signal	Description
clk40M	40MHz clock (2x sample clock)
valid_in	input data valid signal; when valid is low, input data should be ignored. Valid "spread out" to smooth out data rate over whole symbol - average of 1 data valid every six 40MHz cycles. Effective data rate at viterbi input dropped to 20MHz
sdi_data[11:0]	input data from Symbol Deinterleaver. Data is in 6-bit I, 6-bit Q format (for 64_QAM)

Signal	Description
bad_carrier	Set to 1 if a carrier conveying the data fell below cceptable limits; indicates to Viterbi that this data should be ignored
symbol	Timing synchronization signal - high for the first data sample in a COFDM symbol. Used to resynchronize address generation
constellation[1:0]	Constellation Type indicator:10 = 64-QAM01 = 16-QAM00 = QPSK
alpha[2:0]	Hierarchical transmission control:000 = non-hierarchical, 001 = alpha value 1, 010 = alpha value 2, 011 = alpha value 4(Note: in this first version of the device only non-hierarchical mode is supported)

Table 32

Signal	Description
I-data[2:0]	I soft-bit to Viterbi
discard-I	flag bit drived from bad_carrier signal; viterbi will ignore this soft-bit if set.(bad-carrier is repeated per soft-bit because of interleaving)
Q-data[2:0]	Q soft-bit to Viterbi
discard-Q	flag-bit; VIterbi will ignore this soft-bit if set
valid	Valid output data indicator; high if data is being output on this clock cycle

Table 33

Signal	Description
MD[7:0] (bi-di)	Microprocessor Data bus (bi-directional)
MA[5:0]	Microprocessor Address Bus
MR/W	Microprocessor Read / Write control
SCL	Serial Interface Clock
SDA(bi-di)	Serial Interface Data I/O (bi-directional - same pin as MD[0])
SADDR[2:0]	Serial Interface Address
S/P	Serial/Parallel interface select

Table 34

Signal	Description
nupdata[7:0] (bi-di)	Internal processor data bus (inverted) (bi-directional)

Signal	Description
upaddr[5:0]	Internal address bus (decoded to provide individual selects for various register banks within functional blocks)
upgrstr	Internal read strobe
upgwstr	Internal write strobe
IRQ	Interrupt Request (Active low, open collector)

Table 35

Signal	Description
pad_clk40	Uncontrolled 40MHz clock from input pad
lupdata[7:0] (bi-di)	Internal Microprocessor Data bus (bi-directional)
upaddr[3:0]	Internal Microprocessor Address Bus (only bits relevant to registers within System Control)
uprstr	Internal Microprocessor Read strobe
upwstr	Internal Microprocessor Write Strobe
upsel1	block select decoded from microprocessor interface (1 = access to this block enabled) valid for addresses 0x00-0x0b
upsel2	address decode for 0x38-0x39 range
tps_data[10:0]	TPS data received in OFDM frame (1:0 = tps_constellation; 4:2 = tps_alpha7:5 = tps_hp_rate10:8 = tps_lp_rate)(Don't bother with Guard Interval - these parameters only affect back end blocks)
rs_correct[5:0]	Count of bits corrected in each RS packet (accumulated over 1 second for BER value)
rs_corr_val	Valid pulse; high when rs_correct value is valid
pkt_err	Set to 1 to indicate RS packet is uncorrectable; has >64 bit errors or is corrupted in some other way.
err_val	Set to 1 to indicate when pkt_err signal is valid
vit_ill_state	Viterbi_illegal state pulse; (accumulate to give Viterbi illegal state count)
vit_ill_val	NOW NOT REQUIRED - Viterbi illegal state valid pulse
vit_sync	Status signal - 1 if Viterbi is synchronized
tps_sync	Status signal - 1 if TPS is synchronized
pilot_loc	Status signal - 1 if pilot location completed successfully (found_pilots))
fft_loc	Status signal - 1 if FFT window has located correctly
vit_rate[2:0]	Received Viterbi puncture rate.

Signal	Description
tck	JTAG test clock - used for control of clock in test mode
njreset	JTAG test reset - for clock control block
jshift	JTAG test register shift control - for clock control block
j_ctrl_in	JTAG test data input

Table 36

Signal	Description
clk40	Test-controlled main clock
clk20	Test-controlled sample clock (input to IQ Demod and AGC)
iupdata[7:0] (bi-di)	Internal processor data bus (bi-directional)
nirq	Active Low interrupt request bit (derived from chip_event)
constellation[1:0]	Internal address bus (decoded to provide individual selects for various register banks within functional blocks)
alpha[2:0]	Hierarchical mode information
hp_rate[2:0]	Viterbi code rate for High Priority channel (in non-hierarchical mode this is the code rate for the complete channel)
lp_rate[2:0]	Viterbi code rate for Low Priority channel.
upi_tx_mode[1:0]	Transmission mode (2K or 8K)
upi_guard[1:0]	Guard Interval
rxp_valid	Set to 1 if Host Interface has set rx_para data - used as a "valid" signal for rx_para data (in case of TPS data use tps_sync)
o_clk_phase	Control line; set to 1 to invert output clock phase
xtc[2:0]	External Tuner Control bits
i2c_gate	I2C "Gate" control
ts_tri	Transport Stream Interface tristate control
soft_reset	Software Reset (set to 1 to reset everything except upi)
agc_invert	Control line: set to 1 to invert sense of AGC sigma-delta output (default: low output equates to low AGC gain)
agc_resync	Control line: When set low AGC held in initial condition. Resync transitioning high commences the AGC acquisition sequence

Signal	Description
fft_resync	Control line: hold low to re-initialise FFT, Channel Estimation & Correction, Frequency/Sampling Error and TPS blocks. Transition high commences FFT window locate, Pilot locate and TPS synchronisation.
viterbi_resync	Contol line; hold low to re-initiliase FEC backend. Transition high commences Viterbi synchronisation.
j_ctrl_out	JTAG test data output - from clock control block.

Table 37

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
0x00	**Event Reg.**			
	0	R/W/0	chip_event	OR of all events which are interrupt-enabled (unmasked)
	1	R/W/0	lock_failed_event	Set to 1 if channel acquisition sequence fails
	2	R/W/0	rs_overload_event	Set to 1 if Reed-Solomon Decoder exceeds set threshold within one 1 second period
0x01	**Mask Reg.**			
	0	R/W/0	chip_mask	Set to 1 to enable IRQ output
	1	R/W/0	lock_failed_mask	Set to 1 to enable interrupt on channel acquisition fail
	2	R/W/0	rs_overload_mask	Set to 1 to enable interrupt on RS error threshold exceeded
0x02	**Status Reg.**			
	0	R/0	system_locked	Set to 1 when system acquired channel successfully
	1	R/0	viterbi_sync	Set to 1 when Viterbi is synchronized
	2	R/0	tps_sync	Set to 1 when OFDM frame carrying TPS data has been synchronized to.

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	3	R/0	pilot_loc	Set to 1 when pilots in COFDM symbol have been located and synchronized to
	4	R/0	fft_loc	Set to 1 when guard interval has been located and synchronized to.
	7:5	R/1	viterbi_rate	Received Viterbi Code rate
0x04-0x05	Control Reg:			
	0	R/W/0	change_channel	When set to 1, holds device in "Reset" state. Clearing this bit initiates channel change.
	1	R/W/0	agc_invert	Invert AGC Signa-Delta output. Default setting means low output associated with reduced AGC gain.
	2	R/W/0	o_clk_phase	Set to 1 to invert phase of output clock. Default condition: output data changes on falling edge of output clock.
	3	R/W/0	set_rx_parameters	Set to 1 to take Reciver Parameter Data from Receiver Parameter Register. Default condition: settings taken from TPS data (longer channel acquisition time)
	4	R/W/0	extend_agc	Set to 1 to hold acquisition sequence in agc_acquire state
	5	R/W/0	extend_fs	Set to 1 to hold acquisition sequence in fs_acquire state
	6	R/W/0	extend_settle	Set to 1 to hold acquisition sequence in fs_settle state

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	7	R/W/0	extend_syn	When set to 1 to hold acquisition sequence in vit_sync state
	10:8	R/W/0	xtc	External Tuner Control bits (external pins XTC[2:0])
	11	R/W/0	i2c_gate	I2C "Gate" signal: setting this to 1 enables the isolation buffer between the "processor side" I2C bus and the "Tuner side" I2C so the processor can acces a Tuner through COFDM device. Setting to 0 closes the "gate" to prevent I2C bus noise affecting delicate RF.
	12	R/W/0	ts_tri	Transport Stream Tristate control - set to 1 to tristate MPEG TS interface (eg. to mux a QPSK devce to same MPEG demux). Power-on state of TS output controlled by external pin - somehow!!!
	13	R/W/0	fast_ber	Set to 1 to reduce BER counter, vit_ill_state counter and rso_counter, counter periods from 1 sec to 100ms
	15	R/W/0	soft_reset	Software Reset - set to 1 to reset all blocks except upi. Set to 0 to release.
0x06-0x07	Receiver Parameter Register:			
	15:14	R/W/2	upi_constellation	Constellation Pattern for Demapper and Bit Deinterleaver (reset condition = 64-QAM)
	13:12	R/W/0	upi_guard	Guard Interval: 00 = 1/32, 01 = 1/16, 10 = 1/8, 11 = 1/4

Address (Hex)	Bit No.	Dir/Reset	Register Name	Description
	11:9	R/W/0	upi_alpha	Hierarchical Tranmission Mode or "alpha value" (reset condition = non-hierarchical mode)
	7:5	R/W/0	upi_hp_rate	Viterbi Code Rate for HP stream - in non-hierarchical mode this is taken as the Viterbi Code Rate (reset condition = 1/2 rate code)
	4:2	R/W/0	upi_lp_rate	Viterbi Code Rate for LP stream (reset condition = 1/2 rate code)
	1:0	R/W/0	upi_tx_mode	Trnnsmission mode (00=2K, 01=8K, others reserved)
0x08	7:0	R/W/0	rso_limit	Errored packet per second limit (for rs_overload_event bit)
0x09	7:0	R/0	rso_count	Count of Uncorrectable Transport Packets per second (saturates at 255).Write to register to latch a stable count value which can then be read back.
0x0a - 0x0b	15:0	R/0	ber	BER (before RS) deduced from RS corrections in 1 second period - max correctable bit errors ~1.35M/sec for 7/8, 64-QAM, 1/32 GI (equivalent to 43.e-3 BER assuming useful bitrate of 31.67 e6). Only top 16 bits of 21 bit counter are visible - resolution of ~1e-6 depending on code-rate, constellation GI length. Write to register to latch a stable count value which can then be read back.

Table 38

0x38-0x39	11:0	R/0	vit_ill_states	Viterbi Illegal State Rate (per second) Write to register to latch count which can then be read back

Table 39

Listing 1

```
// SccsId: %W% %G%
/*****************************************************************************
    Copyright (c) 1997 Pioneer Digital Design Centre Limited

    Author   : Dawood Alam.

    Description: Verilog code for butterfly processor BF2I. (RTL)

    Notes    : Computes first stage in radix 4 calculation.

*****************************************************************************/

`timescale 1ns / 100ps

module fft_bf2I (clk, enable_1, in_x1r, in_x1i, in_x2r, in_x2i, in_s,
      out_z1r, out_z1i, out_z2r, out_z2i, out_ovf);

parameter      wordlength = 5;     // Data wordlength.

input          clk,          // Master clock.
               enable_1,     // Enable on clock 3.
               in_s;         // Control line.
input [wordlength-1:0] in_x1r,       // Input I from memory.
               in_x1i,       // Input Q from memory.
               in_x2r,       // Input I stage n-1.
               in_x2i;       // Input Q stage n-1.

output         out_ovf;      // Overflow flag.
output [wordlength-1:0] out_z1r,     // Output I to stage n+1
               out_z1i,      // Output Q to stage n+1
               out_z2r,      // Output I to memory.
               out_z2i;      // Output Q to memory.

wire [wordlength-1:0] in_x1r,
               in_x1i,
               in_x2r,
               in_x2i,
               out_z1r,
               out_z1i,
               out_z2r,
               out_z2i;
wire           in_s,
               enable_1,
               out_ovf;

reg [wordlength-1:0] z1r_tmp1,
               z1i_tmp1,
               z2r_tmp1,
               z2i_tmp1,
               z1r_tmp2,
               z1i_tmp2,
               z2r_tmp2,
```

```
            z2i_tmp2
reg         ovf_tmp,
          ovf_tmp0,
          ovf_tmp1,
            ovf_tmp2,
            ovf_tmp3,
          ex_reg0,
            ex_reg1,
            ex_reg2,
            ex_reg3;

always @(in_s or in_x1r or in_x1i or in_x2r or in_x2i)
begin
 {ex_reg0,z1r_tmp1} = in_x1r + in_x2r;
 ovf_tmp0 = in_x1r[wordlength-1] &&          // Overflow check.
     in_x2r[wordlength-1] &&
       ~z1r_tmp1[wordlength-1] ||
     ~in_x1r[wordlength-1] &&
       ~in_x2r[wordlength-1] &&
       z1r_tmp1[wordlength-1];
 if (ovf_tmp0)                  // Saturate logic.
  z1r_tmp1 = (ex_reg0) ? {1'b1,{wordlength-1{1'b0}}} :
          {1'b0,{wordlength-1{1'b1}}};

 {ex_reg1,z1i_tmp1} = in_x1i + in_x2i;
 ovf_tmp1 = in_x1i[wordlength-1] &&          // Overflow check.
     in_x2i[wordlength-1] &&
     ~z1i_tmp1[wordlength-1] ||
     ~in_x1i[wordlength-1] &&
       ~in_x2i[wordlength-1] &&
       z1i_tmp1[wordlength-1];
 if (ovf_tmp1)                  // Saturate logic.
  z1i_tmp1 = (ex_reg1) ? {1'b1,{wordlength-1{1'b0}}} :
          {1'b0,{wordlength-1{1'b1}}};

 {ex_reg2,z2r_tmp1} = in_x1r - in_x2r;
 ovf_tmp2 = in_x1r[wordlength-1] &&          // Overflow check.
     ~in_x2r[wordlength-1] &&
       ~z2r_tmp1[wordlength-1] ||
     ~in_x1r[wordlength-1] &&
       in_x2r[wordlength-1] &&
     z2r_tmp1[wordlength-1];
 if (ovf_tmp2)                  // Saturate logic.
  z2r_tmp1 = (ex_reg2) ? {1'b1,{wordlength-1{1'b0}}} :
          {1'b0,{wordlength-1{1'b1}}};

 {ex_reg3,z2i_tmp1} = in_x1i - in_x2i;
 ovf_tmp3 = in_x1i[wordlength-1] &&          // Overflow check.
     ~in_x2i[wordlength-1] &&
     ~z2i_tmp1[wordlength-1] ||
     ~in_x1i[wordlength-1] &&
     in_x2i[wordlength-1] &&
       z2i_tmp1[wordlength-1];
 if (ovf_tmp3)                  // Saturate logic.
  z2i_tmp1 = (ex_reg3) ? {1'b1,{wordlength-1{1'b0}}} :
          {1'b0,{wordlength-1{1'b1}}};
```

```
    // Output stage with two channel mux.
    if (!in_s)
     begin: mux_passthru
        z1r_tmp2 = in_x1r;
        z1i_tmp2 = in_x1i;
        z2r_tmp2 = in_x2r;
        z2i_tmp2 = in_x2i;
     end
    else
     begin: mux_computing
        z1r_tmp2 = z1r_tmp1;
        z1i_tmp2 = z1i_tmp1;
        z2r_tmp2 = z2r_tmp1;
        z2i_tmp2 = z2i_tmp1;
     end
    end

    assign out_z1r = z1r_tmp2;
    assign out_z1i = z1i_tmp2;
    assign out_z2r = z2r_tmp2;
    assign out_z2i = z2i_tmp2;

    always @(posedge clk)
     if (enable_1)     // Butterfly completes at the end of clock cycle 0.
      ovf_tmp <= in_s && (ovf_tmp0 || ovf_tmp1 || ovf_tmp2 || ovf_tmp3);

    assign out_ovf = ovf_tmp;

    `ifdef OVERFLOW_DEBUG_LOW_LEVEL
    // Debug code to display overflow output of a particular adder.
    // Concurrently monitor overflow flag and halt on overflow.
    always @(ovf_tmp or ovf_tmp0 or ovf_tmp1 or ovf_tmp2 or ovf_tmp3)
     if (ovf_tmp)
      begin
        if (ovf_tmp0) $display("ovf_tmp0 on BF2I = ",ovf_tmp0);
        if (ovf_tmp1) $display("ovf_tmp1 on BF2I = ",ovf_tmp1);
        if (ovf_tmp2) $display("ovf_tmp2 on BF2I = ",ovf_tmp2);
        if (ovf_tmp3) $display("ovf_tmp3 on BF2I = ",ovf_tmp3);
        $stop;
      end
    `endif
    endmodule
```

Listing 2

```
    // SccsId: %W% %G%
    /*****************************************************************************
        Copyright (c) 1997 Pioneer Digital Design Centre Limited

    Author   : Dawood Alam.

    Description: Verilog code for butterfly processor BF2II. (RTL)

    Notes    : Computes second stage in radix 4 calculation.
```

```
*****************************************************************************/

`timescale 1ns / 100ps

module fft_bf2II (clk, enable_1, in_x1r, in_x1i, in_x2r, in_x2i, in_s, in_t,
      out_z1r, out_z1i, out_z2r, out_z2i, out_ovf);

parameter      wordlength = 5;      // Data wordlength.

input        clk,          // Master clock.
         enable_1,        // Enable on clock 3.
           in_s,          // Control line.
           in_t;          // Control line.
input [wordlength-1:0] in_x1r,         // Input I from memory.
         in_x1i,         // Input Q from memory.
           in_x2r,        // Input I stage n-1.
           in_x2i;        // Input Q stage n-1.

output       out_ovf;         // Overflow flag.
output [wordlength-1:0] out_z1r,         // Output I to stage n+1
         out_z1i,         // Output Q to stage n+1
           out_z2r,       // Output I to memory.
           out_z2i;       // Output Q to memory.

wire  [wordlength-1:0] in_x1r,
         in_x1i,
           in_x2r,
           in_x2i,
         out_z1r,
           out_z1i,
           out_z2r,
           out_z2i;
wire        in_s,
         in_t,
           enable_1,
           out_ovf,
           control;

reg  [wordlength-1:0] z1r_tmp1,
         z1i_tmp1,
           z2r_tmp1,
           z2i_tmp1,
         z1r_tmp2,
           z1i_tmp2,
           z2r_tmp2,
           z2i_tmp2,
         x2ri_tmp1,
           x2ri_tmp2;
reg         ovf_tmp,
         ovf_tmp0,
         ovf_tmp1,
           ovf_tmp2,
           ovf_tmp3,
         ex_reg0,
           ex_reg1,
           ex_reg2,
```

```
            ex_reg3;

    assign control = in_s && !in_t;

    always @(in_s or control or in_x1r or in_x1i or in_x2r or in_x2i)
    begin
     // Crosspoint switch, used in computing complex j values.
     if (control)
      begin: switch_crossed
       x2ri_tmp1 = in_x2i; // i -> r.
       x2ri_tmp2 = in_x2r; // r -> i.
      end
     else
      begin: switch_thru
       x2ri_tmp1 = in_x2r; // r -> r.
       x2ri_tmp2 = in_x2i; // i -> i.
      end

    {ex_reg0,z1r_tmp1} = in_x1r + x2ri_tmp1;
    ovf_tmp0 = in_x1r[wordlength-1] &&          // Overflow check.
        x2ri_tmp1[wordlength-1] &&
          ~z1r_tmp1[wordlength-1] ||
        ~in_x1r[wordlength-1] &&
          ~x2ri_tmp1[wordlength-1] &&
          z1r_tmp1[wordlength-1];
    if (ovf_tmp0)                   // Saturate logic.
     z1r_tmp1 = (ex_reg0) ? {1'b1,{wordlength-1{1'b0}}} :
              {1'b0,{wordlength-1{1'b1}}};

    {ex_reg1,z1i_tmp1} = (control) ? in_x1i - x2ri_tmp2:in_x1i + x2ri_tmp2;
    ovf_tmp1 = in_x1i[wordlength-1] &&          // Overflow check.
        (control ^ x2ri_tmp2[wordlength-1]) &&   // Deals with a
          ~z1i_tmp1[wordlength-1] ||         // +/- input.
        ~in_x1i[wordlength-1] &&
          ~(control ^ x2ri_tmp2[wordlength-1]) &&
          z1i_tmp1[wordlength-1];
    if (ovf_tmp1)                   // Saturate logic.
     z1i_tmp1 = (ex_reg1) ? {1'b1,{wordlength-1{1'b0}}} :
              {1'b0,{wordlength-1{1'b1}}};

    {ex_reg2,z2r_tmp1} = in_x1r - x2ri_tmp1;
    ovf_tmp2 = in_x1r[wordlength-1] &&          // Overflow check.
          ~x2ri_tmp1[wordlength-1] &&       // Deals with a
          ~z2r_tmp1[wordlength-1] ||        // - input.
        ~in_x1r[wordlength-1] &&
          x2ri_tmp1[wordlength-1] &&
          z2r_tmp1[wordlength-1];
    if (ovf_tmp2)                   // Saturate logic.
     z2r_tmp1 = (ex_reg2) ? {1'b1,{wordlength-1{1'b0}}} :
              {1'b0,{wordlength-1{1'b1}}};

    {ex_reg3,z2i_tmp1} = (control) ? in_x1i + x2ri_tmp2:in_x1i - x2ri_tmp2;
    ovf_tmp3 = in_x1i[wordlength-1] &&          // Overflow check.
        ~(control ^ x2ri_tmp2[wordlength-1]) &&   // Deals with a
          ~z2i_tmp1[wordlength-1] ||         // -/+ input.
        ~in_x1i[wordlength-1] &&
```

```
        (control ^ x2ri_tmp2[wordlength-1]) &&
        z2i_tmp1[wordlength-1];
      if (ovf_tmp3)                    // Saturate logic.
       z2i_tmp1 = (ex_reg3) ? {1'b1,{wordlength-1{1'b0}}} :
            {1'b0,{wordlength-1{1'b1}}};

      // Output stage with two channel mux.
      if (!in_s)
       begin: mux_passthru
         z1r_tmp2 = in_x1r;
         z1i_tmp2 = in_x1i;
         z2r_tmp2 = x2ri_tmp1;
         z2i_tmp2 = x2ri_tmp2;
       end
      else
       begin: mux_computing
         z1r_tmp2 = z1r_tmp1;
         z1i_tmp2 = z1i_tmp1;
         z2r_tmp2 = z2r_tmp1;
         z2i_tmp2 = z2i_tmp1;
       end
      end

      assign out_z1r = z1r_tmp2;
      assign out_z1i = z1i_tmp2;
      assign out_z2r = z2r_tmp2;
      assign out_z2i = z2i_tmp2;


      always @(posedge clk)
       if (enable_1)     // Butterfly completes at the end of clock cycle 0.
        ovf_tmp <= in_s && (ovf_tmp0 || ovf_tmp1 || ovf_tmp2 || ovf_tmp3);

      assign out_ovf = ovf_tmp;


      `ifdef OVERFLOW_DEBUG_LOW_LEVEL
      // Debug code to display overflow output of a particular adder.
      // Concurrently monitor overflow flag and halt on overflow.
      always @(ovf_tmp or ovf_tmp0 or ovf_tmp1 or ovf_tmp2 or ovf_tmp3)
       if (ovf_tmp)
        begin
          if (ovf_tmp0) $display("ovf_tmp0 on BF2II = ",ovf_tmp0);
          if (ovf_tmp1) $display("ovf_tmp1 on BF2II = ",ovf_tmp1);
          if (ovf_tmp2) $display("ovf_tmp2 on BF2II = ",ovf_tmp2);
          if (ovf_tmp3) $display("ovf_tmp3 on BF2II = ",ovf_tmp3);
          $stop;
        end
       `endif
      endmodule
```

Listing 3

```
// SccsId: %W% %G%
/*****************************************************************************
```

```
Author   : Dawood Alam.

Description: Verilog code for a variable size ROM with complex data store.
    (RTL)

Notes    : Used to store complex Twiddle factors.

*****************************************************************************/

`timescale 1ns / 100ps

module fft_rom (clk, enable_3, address, rom_data);

parameter       c_wordlength = 1;    // Coeff wordlength.
parameter       rom_AddressSize = 1;  // Address size.
parameter FILE = "../../../fft/src/lookup_tables/lu_10bit_2048pt_scaleX";
                // Lookup tab filename. (Listings 16, 17)

input          clk,
               enable_3;
input [rom_AddressSize-1:0] address;

output [c_wordlength-1:0]  rom_data;

reg  [c_wordlength*2-1:0] rom [0:(1 << rom_AddressSize)-1];
reg  [c_wordlength*2-1:0] b_tmp1,
               rom_data;

always @(address)
 b_tmp1 = rom[address];

always @(posedge clk)
 if (enable_3)
  rom_data <= b_tmp1;

initial
 $readmemb(FILE, rom);

endmodule
```

Listing 4

```
// SccsId: %W% %G%
/*****************************************************************************
    Copyright (c) 1997 Pioneer Digital Design Centre Limited

Author   : Dawood Alam.

Description: Verilog code for variable length single bit shift register.

Notes    : Used to delay pipeline control signals by "length" clocks.

*****************************************************************************/

`timescale 1ns / 100ps
```

```
module fft_sr_1bit (clk, enable_3, in_data, out_data);

 parameter       length = 1;       // Shift reg length.

 input         clk,          // Master clock;
             enable_3;       // Enable on clock 3.
 input         in_data;        // Input data.

 output          out_data;       // Output data.

 reg           shift_reg [length-1:0]; // Shift register.

 wire          out_data;
 wire          clk;
             enable_3;
 integer         i;

 always @ (posedge clk)
  if (enable_3)
   begin
     for (i = (length-1); i >= 0; i = i - 1)
      if (i == 0)
       shift_reg[0] <= in_data;        // Force input to SR.
      else
       shift_reg[i] <= shift_reg[i-1];     // Shift data once.
   end

 assign out_data = shift_reg[length-1];
endmodule
```

Listing 5

```
// SccsId: %W% %G%
/*****************************************************************************
    Copyright (c) 1997 Pioneer Digital Design Centre Limited

 Author   : Dawood Alam.

 Description: Verilog code for a dual-port FIFO. (RTL)

 Notes    : Used as a pipeline register to delay address into the address
     decoder.

*****************************************************************************/

`timescale 1ns / 100ps

module fft_sr_addr (clk, enable_3, in_data, out_data);

 parameter       wordlength = 1;     // Data wordlength I/Q.
 parameter       length = 1;       // Shift reg length.

 input         clk,          // Master clock;
             enable_3;       // Enable on clock 3.
 input  [wordlength-1:0]  in_data;        // SR input data.
 output [wordlength-1:0]  out_data;       // SR output data.
```

```
reg  [wordlength-1:0]  shift_reg [length-1:0];  // Shift register.

wire [wordlength-1:0]  out_data;
wire          clk,
              enable_3;
integer       i;

always @ (posedge clk)
 if (enable_3)
  begin
   for (i = (length-1); i >= 0; i = i - 1)
     if (i == 0)
      shift_reg[0] <= in_data;        // Force input to SR.
    else
      shift_reg[i] <= shift_reg[i-1];     // Shift data once.
  end

assign out_data = shift_reg[length-1];
endmodule
```

Listing 6

```
// SccsId: %W% %G%
/* Copyright (c) 1997 Pioneer Digital Design Centre Ltd.

 Author   : Dawood Alam.

 Description: Verilog code for an signed twiddle factor multiplier. (RTL)

 Notes   : Single multiplexed multiplier and 2 adders employed to
     perform 3 multiplies and 5 additions. Pipeline depth = 2.
     ar/ai = Complex data, br/bi = Complex coefficient.
     bi +/- br could be pre-calculated in the ROM lookup, however
     in this implementation it is NOT an overhead as this path is
        shared by ar + ai. */

`timescale 1ns / 100ps

module fft_complex_mult_mux (clk, c2, in_ar, in_ai, in_br, in_bi,
                  out_cr, out_ci, out_ovf);

parameter      wordlength = 12;    // Data wordlength.
parameter      c_wordlength = 10;   // Coeff wordlength.
parameter      mult_scale = 4;     // multiplier scalling,
                 // 1 = /4096, 2 = /2048,
                 // 3 = /1024, 4 = /512.

input [wordlength-1:0]  in_ar,        // Data input I.
          in_ai;        // Data input Q.
input [c_wordlength-1:0] in_br,        // Coefficient input I.
          in_bi;        // Coefficient input Q.
input         clk;          // Master clock.
input [1:0]      c2;           // Two bit count line.
output        out_ovf;        // Overflow flag.
output [wordlength-1:0]  out_cr,        // Data output I.
          out_ci;        // Data output Q.
```

```
        wire  [wordlength-1:0]  in_ar,
                    in_ai,
                      br_tmp,
                      bi_tmp,
                      out_cr,
                      out_ci;
        wire  [c_wordlength-1:0] in_br,
                    in_bi;
        wire          enable_0,
                      enable_1,
                      enable_2,
                      enable_3;
        wire  [1:0]       c2;

        reg   [wordlength-1:0]  in_ai_tmp,
                    in_ar_tmp,
                      abr_tmp,
                    abi_tmp,
                      abri_tmp1,
                      abri_tmp2,
                      abri_tmp4,
                      coeff_tmp1,
                    mpy_tmp1,
                      sum_tmp0,
                      sum_tmp1,
                      sum_tmp2,
                      acc_tmp,
                      store_tmp,
                      cr_tmp,
                      ci_tmp;
        reg   [wordlength*2-1:0] abri_tmp3,
                    mpy_tmp2,
                      coeff_tmp2;
        reg           ovf_tmp0,
                    ovf_tmp1,
                      ovf_tmp2,
                      ovf_tmp3,
                    ex_reg0,
                      ex_reg1,
                      c1, c3, c4;

        // Enable signals for registers.
        assign enable_0 = ~c2[1] && ~c2[0];
        assign enable_1 = ~c2[1] && c2[0];
        assign enable_2 =  c2[1] && ~c2[0];
        assign enable_3 =  c2[1] && c2[0];

        // Sign extend coefficients from c_wordlength bits to wordlength.
        assign br_tmp = {{(wordlength-c_wordlength){in_br[c_wordlength-1]}},in_br};
        assign bi_tmp = {{(wordlength-c_wordlength){in_bi[c_wordlength-1]}},in_bi};

        // Combinational logic before pipeline register.
        always @(in_ar or br_tmp or in_ai or bi_tmp or c2)
          begin
```

```
        c1 = c2[0] || c2[1];
        c3 = c2[1];

        if (!c1)
          begin
           abr_tmp = in_ar;
           abi_tmp = in_ai;
          end
        else
          begin
           abr_tmp = br_tmp;
           abi_tmp = bi_tmp;
          end

        if (c3)
          {ex_reg0,abri_tmp4} = abi_tmp - abr_tmp;
        else
          {ex_reg0,abri_tmp4} = abi_tmp + abr_tmp;

        ovf_tmp0 = abi_tmp[wordlength-1] &&        // Overflow check.
            (c3 ^ abr_tmp[wordlength-1]) &&      // Deals with a
            ~abri_tmp4[wordlength-1] ||          // +/- input.
            ~abi_tmp[wordlength-1] &&
            ~(c3 ^ abr_tmp[wordlength-1]) &&
            abri_tmp4[wordlength-1];

        if (ovf_tmp0)                  // Saturate logic.
          abri_tmp1 = (ex_reg0) ? {1'b1,{wordlength-1{1'b0}}} :
                {1'b0,{wordlength-1{1'b1}}};
        else
          abri_tmp1 = abri_tmp4;

       end

      // Combinational logic after pipeline register.
      always @(in_ar_tmp or in_ai_tmp or br_tmp or c2 or store_tmp or abri_tmp2)
       begin
       c4 = c2[1] && ~c2[0];

       case (c2)
       2'b00:
        begin
          coeff_tmp1 = in_ar_tmp;
          sum_tmp0 = store_tmp;
         end
       2'b01:
        begin
          coeff_tmp1 = br_tmp;
          sum_tmp0 = {wordlength-1{1'b0}};
         end
       2'b10:
        begin
          coeff_tmp1 = in_ai_tmp;
          sum_tmp0 = store_tmp;
         end
       2'b11:
```